ספר התשובות לפסקל

תשובות לתרגילי התכנות שבספר
פסקל — שפה לתכנות מבני
מאת **פרופ' אבא אנגלברג**

אמיר קפלן

ערך **זהר עמיהוד**

- ★ **200 תוכניות למתחיל ולמתקדם, עם הסברים ותרשימים סכימטיים**
  - פונקציות ופרוצדורות
  - תוכנית כלליות למיון ומיזוג
  - תוכניות QuickSort ו־HeapSort
  - רשומות וקבצים
  - קבצי תמליל (Text Files)
  - רקורסיות
  - מבני נתונים דינמיים
- ★ **תרשימי מבנה לשפת פסקל (Syntax Diagrams)**
- ★ **Quick Reference להגדרת מהדר פסקל הסטנדרטי ומהדר Turbo-Pascal 3.0**

ספרי לימוד והכשרה במדעי המחשב
הוצאת הוד־עמי לספרי מחשבים

הוצאת הוד-עמי
לספרי מחשבים בע"מ

ת.ד. 560 רמת-גן, 52105
טל' 052-541207

12 במרץ, 1987

לכב'
משה מלמד
יועץ לענייני ענ"א

רח' הרצוג 21
גבעתיים

שלום,

אני מגיש לעיונך את הספר החדש, אשר הוצאנו לאור זה עתה,

ספר התשובות לפסקל
מאת אמיר קפלן

תשובות לתרגילי התכנות שבספר

פסקל — שפה לתכנות מבני
מאת פרופ' א' אנגלברג

ספר זה מצטרף לספרים אחרים בהוצאתנו, אשר התקבלו כספרי עזר וכספרי לימוד במוסדות חינוך רבים בארץ.

ספר התשובות לפסקל מציג בפני הקורא מיגוון של תוכניות העוסקות במספר רב של נושאים למתחילים ולמתקדמים. אלו הן תשובות לשאלות החזרה ותרגילי התכנות שבספר "פסקל - שפה לתכנות מבני", שבהוצאתנו.

ניתן דגש לנושאים: מיון, רשימות ומצביעים, קבצים ומבני נתונים דינמיים. התוכניות הורצו במחשב IBM PC ומהדר Turbo Pascal 3.0. לספר נספחים הכוללים תרשימי מבנה והגדרות של מהדר פסקל. אינדקס הנושאים ממויין בעברית ובאנגלית.

הספר מותאם לדרישות תוכניות הלימודים החדשה של משרד החינוך והתרבות, עבור מגמות ענ"א בבתי הספר העל-יסודיים ובמכללות להנדסאים. הוא מתאים לקורסים במרכז ענ"א במכון לפריון העבודה והייצור, המכון להכשרה טכנולוגית (מה"ט) ולתוכניות הלימודים באוניברסיטאות ובמכללות למחשבים.

אני מקווה שהספר יקבל את תשומת הלב המתאימה וישמש כספר לימוד, כמו ספרים אחרים שבהוצאתנו.

בברכה,

יצחק עמיהוד

נ.ב.
לעיונך בלבד...

ספר התשובות
לפסקל

ערב פסח תשמ"ז
1987

מוקדש באהבה ובהוקרה

להורי

משה ראב כתב את נספח ב'.

הסדר והעריכה נעשו במחשב IBM PC באמצעות מעבד התמלילים

א-ב 4.10

של חברת גרינברג מהנדסים בע"מ.

הדפסת הספר נעשתה במדפסת Canon LBP-8 A1
המשווקת ע"י קרט בע"מ.

ספר התשובות לפסקל

תשובות לתרגילי התכנות שבספר
פסקל — שפה לתכנות מבני
מאת פרופ׳ אבא אנגלברג

אמיר קפלן

ערך **זהר עמיהוד**

ספרי לימוד והכשרה במדעי המחשב
הוצאת הוד־עמי לספרי מחשבים

PASCAL TRAINING

by
Amir Kaplan
edited by Z. Amihud

Answers to
Programming Problems
in

P A S C A L

A Language for Structured Programming
by
Prof. A. Engelberg

הוצאת הוד עמי
לספרי מחשבים בע"מ

ת.ד. 560 רמת-גן 52105
טל. 052-541207

הודפס בישראל
שבט תשמ"ז, פברואר 1987

Hod-Ami Ltd
P.O.Box 560 Ramat-Gan 52105
Israel, Feb. 1987

הקדמה

<u>על הספר</u>

ספר התשובות לפסקל מציג בפני הקורא מיגוון של תוכניות פסקל העוסקות במספר רב של נושאים למתחילים ומתקדמים. התשובות ערוכות בפרקים, והן מתייחסות לשאלות החזרה ותרגילי התכנות שבספר "<u>פסקל - שפה לתכנות מבני</u>" מאת פרופ' אבא אנגלברג (בכל מקום שיוזכר ספר פסקל, הכוונה היא לספר זה), בהוצאת הוד-עמי.

השאלות מאפשרות לתלמיד לבחון את ידיעותיו, ותרגילי התכנות מכוונים להכשיר אותו לפתור בעיות מעשיות בשפת התכנות.

ספר התשובות לפסקל מתרגל את הלומד ע"י חשיפה לשיטת התכנות המבני של פסקל, ומדריך אותו לתכנת כראוי, הן מבחינת מבנה התוכנית והן מבחינת התיעוד שלה. במיוחד הושם דגש על הצגת הדרך הלוגית של הפתרון ועל ההסברים לתוכניות, כדי שהלומד יוכל להשתמש בכלים הניתנים בו, לפיתוח יישומים חדשים. בכך, יש לראות אותו כהמשך לנושאי הלימוד בדרך של שינון, חזרה והרחבה של הנושאים שנלמדו.

הפתרונות הניתנים בספר מלווים בהסברים מפורטים ותרשימים סכימטיים של מהלך הביצוע של התוכניות, הסבר למשתנים, תיאור הפרוצדורות והפונקציות ותפקידן, מבנה המערכים לסוגיהם, הקבצים והרשימות המקושרות. התשובות לתרגילי **<u>המיון</u>, <u>הרשימות</u>, <u>הקבצים</u> <u>ומבני נתונים דינמיים</u>** כוללות דוגמאות רבות ונדונו בהרחבה.

מומלץ לפתור תחילה את תרגילי התכנות על פי רמת הידע בכל שלב לימוד. אח"כ - לעיין בתשובה וללמוד את הגישה לפתרון הבעיה, בחירת הכלים לפתרון, ההגדרות, התכנות והתיעוד. במהלך הצגת הפתרונות לבעיות הודגמו פרוצדורות, פונקציות ותוכניות שרות שונות, אשר ניתן להשתמש בהן בעבודה מעשית.

התוכניות הורצו במחשב IBM-PC עם מערכת הפעלה DOS 3.2 ומהדר **Turbo-Pascal 3.0**. מספר תוכניות הורצו במהדר פסקל במחשב VAX (בגלל מגבלות של מהדר Turbo בפקודות מסוימות).

לפרק המיון מצורפות תוכניות המיון **QuickSort** ו- **HeapSort**, בנוסף לתוכניות מיון בועות, מיון ע"י שיבוץ ואחרות, שניתנו בפתרונות לתרגילים. פרק זה מלווה בהסברים מקיפים ודוגמאות.

נספח א' - **Syntax Diagrams** - מציג את תרשימי המבנה של המשפטים בשפת פסקל הסטנדרטית. נספח זה נכתב באנגלית כמקובל בכתיבת תוכניות בפסקל.

נספח ב' הינו **Quick Reference מקיף** (27 רשימות), אשר מתאר את ההגדרות של מהדר פסקל הסטנדרטי ושל המהדר **Turbo-Pascal 3.0**. כל מושג, הגדרה, פונקציה, פרוצדורה, סימן וכו' מלווים בהסבר, אשר מאפשר ברוב המקרים את השימוש בהם ללא צורך במקורות נוספים.

אינדקס הנושאים מאפשר למשתמש לפנות באופן ישיר אל התוכניות העוסקות בנושא אשר דרוש לו, כמו: מיון לסוגיו השונים, רקורסיה, מערכים, מחסניות, מבני נתונים וכדומה. באינדקס משולב גם מילון, אשר ממויין מיון כפול - לפי העברית ולפי האנגלית.

הספר מותאם לדרישות של תוכנית הלימודים החדשה של משרד החינוך והתרבות עבור מגמות ענ"א בבתי הספר העל-יסודיים ובמכללות להנדסאים. הוא מתאים לקורסים במרכז ענ"א במכון לפריון העבודה והייצור, המכון להכשרה טכנולוגית (מה"ט) ולתוכניות הלימודים באוניברסיטאות ובמכללות למחשבים.

הספר מיועד לתלמיד בכתה, ללומד בעצמו ולכל מי שכבר מתכנת בפסקל ומעוניין להעמיק את ידיעותיו.

פברואר 87 אמיר קפלן זהר עמיהוד

תוכן העניינים

פרק 1
תכניות חישוב והדפסה

שאלות

שאלה 2

להלן שמות המשתנים שאינם חוקיים:

א. $DOLLAR - מתחיל ב- $.
ד. 100ACHUZ - מתחיל בספרה.
ו. X-RAY - התו "-" אינו חוקי בשם.

שאלה 3

```
VAR
   PARTNO       : INTEGER
   PRICE, VAT, PROFIT : REAL;
```

שאלה 4

רשימת השגויים:

ב. 4,587 - התו "," אינו חוקי במספר.
ד. .8376 - בהצגת מספר ממשי תופיע ספרה אחת לפחות משמאל לנקודה.
ט. 28937265 - יכול להיות שתהיה גלישה (תלוי במחשב).
יג. 1.7E999 - הערך מימין ל-E בגבולות 38+ עד 39-.
יד. 132. - בהצגת מספר ממשי תהיה ספרה אחת לפחות מימין לנקודה.

שאלה 5

רשימת הפעולות השגויות:

ד. Z:=X DIV C - מתבצעת חלוקה בשלמים (C) על משתנה ממשי (X).
ו. B:=X+Y - החיבור של X ו- Y הוא מספר ממשי, לעומת B שהינו מספר שלם.
ח. A:=Z - השמה של משתנה ממשי לתוך משתנה המוגדר שלם.

שאלה 6

א. (X+2)/(Y+4)
ב. Y+3/(Y+4)
ג. X*Y/4+Y
ד. (X-4)/(2+Y/3)

שאלה 7

```
CONST
     VALUE = 435.59
     LEVEL = 132
     STR   = 'HI THERE'
```

שאלה 8

א. C=6E2 B=4.0 A=-3.0

ב. C=-769E4 Y=199 B=6E2 X=4 A=-3.0

ג. C=6E2 B=4.0 A=-3.0 - שורה ראשונה.
C=62.0 B=8.7 A=-769E4 - שורה אחרונה.

שאלה 9

א. THE VALUE OF 5 =^^^^^^^^5.0E00

ב. THE SUM OF 2+3=^^^^^^^^^5

ג. 1234-345.67
^^1234
^-345

התו ^ מציין תו רווח (blank).

<u>שים לב:</u> אורך שדה ההדפסה של מספר לפי ברירת המחדל, שונה ממחשב למחשב. במחשב IBM-PC אורך שדה ההדפסה למספר ממשי - 14 תווים, ולמספר שלם - 10 תווים.

תוכניות

תוכנית 1

```
(1)  PROGRAM HIGH (INPUT, OUTPUT);
(2)  CONST
(3)     FACTOR = 2.54;
(4)  VAR
(5)     HEIGTH_CM, HEIGTH_INC : REAL;
(6)  BEGIN
(7)     WRITE ('PLEASE ENTER HEIGTH IN INCHES : ');
(8)     READLN (HEIGTH_INC);                   { קרא את הגובה באינצ'ים }
(9)     HEIGHT_CM := FACTOR * HEIGTH_INC;      { חישוב הגובה בסנטימטרים }
(10)    WRITELN ('THE HEIGHT IS ', HIEGHT_CM:6:2, 'IN CENTIMETERS')
(11) END.
```

<u>הסבר:</u>

1 שם התוכנית.
2-3 הגדרת קבוע FACTOR, שהוא יחס המעבר בין אינצ'ים לסנטימטרים.
4-5 הגדרת משתנים.
7-8 קלט - גובה באינצ'ים.
9 חישוב הגובה בסנטימטרים.
10 פלט.

תוכנית 2

```
PROGRAM CHANGE (INPUT, OUTPUT);
VAR
   SHEKALIM, ACHUZ, SHAAR, DOLLARS : REAL;
BEGIN
   WRITE ('HOW MANY SHEKELS DO YOU HAVE? ');
   READLN (SHEKALIM);                        { קרא את סכום השקלים }
   WRITE ('ACHUZ? ');
   READLN (ACHUZ);                           { קרא את אחוז העמלה }
   WRITE ('SHAAR? ');
   READLN (SHAAR);                           { קרא את שער הדולר }
   DOLLARS := SHEKALIM * ( 1 - ACHUZ / 100 ) / SHAAR;
   WRITELN ('YOU GET ', DOLLARS:8:2, ' DOLLARS')
END.
```

<u>הערה:</u>

מספר הדולרים יחושב בדרך הבאה: מספר השקלים לאחר ניכוי העמלה, מחולק בשער הדולר.

תוכנית 3

```
PROGRAM EX_MASKORET2 (INPUT, OUTPUT);
CONST
   MATANA = 25;
   MAS = 0.053;  { 5.3% }
VAR
   SHIUR,
   SHAOT : REAL;
BEGIN
   WRITELN ('MAHU SHIUR HASACHAR? KAMA SHAOT HU AVAD?');
   READLN (SHIUR,SHAOT);
   WRITELN (' שעות   שכר            סה"כ          מס     יתרה');
   WRITELN ('לשעה עבודה   מתנה   תשלומים      תשלום הכנסה');
   WRITELN ('---------------------------------------------');
   WRITELN ('  ', ROUND(SHAOT):3, ' ', SHIUR:6:3, ' ', MATANA:6,
       ' ',SHAOT * SHIUR + MATANA:8:3, ' ', MAS * SHAOT * SHIUR:8:3,
       ' ', (1-MAS) * SHAOT * SHIUR + MATANA:8:3)
END.
```

תוכנית 4

```
PROGRAM RECTANGLE (INPUT, OUTPUT);
VAR
   LENGTH, WIDTH,
   AREA, PERIMETER, DIAGONAL : REAL;
BEGIN
   WRITE ('ENTER RECTANGLE LENGTH AND WIDTH : ');
   READLN (LENGTH, WIDTH);              { קרא נתוני אורך ורוחב של מרובע }
   AREA := LENGTH * WIDTH;                    { חישוב שטח של מרובע }
   PERIMETER := 2 * (LENGTH + WIDTH);         { חישוב היקף המרובע }
   DIAGONAL := SQRT (SQR (LENGTH) + SQR (WIDTH));       { פיתגורס }
   WRITE ('THE LENGTH OF THE RECTANGLE IS ', LENGTH:3:1);
   WRITELN (' ITS WIDTH IS ', WIDTH:3:1);
   WRITELN (' THE AREA IS ', AREA:3:1, ' THE PERIMETER IS ',
       PERIMETER:3:1, ' AND THE DIAGONAL LENGTH IS ', DIAGONAL:3:1)
END.
```

(היקף = PERIMETER)

תוכנית 5

```
PROGRAM TRIANGLE (INPUT, OUTPUT);
VAR
   BASE,
   HYPOTENUSE,                                        { יתר במשולש ישר זוית }
   HEIGHT, SINUS, COSINUS, TANGENS : REAL;
BEGIN
   WRITE ('ENTER TRIANGLE BASE AND HYPOTENUSE : ');
   READLN (BASE, HYPOTENUSE);                   { קרא בסיס ויתר של משולש י"ז }
   HEIGHT  := SQRT (SQR (HYPOTENUSE) - SQR (BASE));              { פיתגורס }
   SINUS   := HEIGHT / HYPOTENUSE;                               {   סינוס }
   COSINUS := BASE / HYPOTENUSE;                                 { קוסינוס }
   TANGENS := SINUS / COSINUS;                                   {   טנגנס }
   WRITELN ('THE TRIANGLE BASE IS ', BASE:3:1,
     ' AND ITS HYPOTENUSE LENGTH IS ', HYPOTENUSE:3:1);
   WRITELN (' ITS HEIGHT IS ', HEIGHT:3:1);
   WRITELN (
     'THE SINUS OF THE ANGLE BETWEEN THE BASE AND THE HYPOTENUSE',
     ' IS', SINUS:3:1);
   WRITELN ('THE COSINUS IS ', COSINUS:3:1);
   WRITELN ('AND THE TANGENS IS ', TANGENS:3:1)
END.
```

תוכנית 6

```
PROGRAM BOX (INPUT, OUTPUT);
CONST
   PRICE = 30;
VAR
   BOX_LENGTH,
   BOX_WIDTH,
   BOX_HEIGHT,
   SIDE_WALL_AREA,
   BASE_AREA,
   FRONT_WALL_AREA,
   VOLUME,
   COST          : REAL;
BEGIN
   WRITE('PLEASE ENTER BOX LENGTH : ');
   READLN (BOX_LENGTH);
   WRITE ('BOX WIDTH : ':26);
   READLN (BOX_WIDTH);
   WRITE ('BOX HEIGHT : ':26);
   READLN (BOX_HEIGHT);
   SIDE_WALL_AREA := BOX_LENGTH * BOX_HEIGHT; {  שטח דופן צדדית }
   BASE_AREA := BOX_LENGTH * BOX_WIDTH;       {  שטח בסיס התיבה }
   FRONT_WALL_AREA := BOX_WIDTH * BOX_HEIGHT; { שטח דופן קידמית }
```

```
    VOLUME := BASE_AREA * BOX_HEIGHT;                          { נפח }
    COST := 2 * ( BASE_AREA + SIDE_WALL_AREA +
             FRONT_WALL_AREA ) * PRICE / 10000;                { עלות }
    WRITELN;
    WRITELN ('FOR THIS BOX SIDE WALL AREA IS : ',
            SIDE_WALL_AREA:8:2, '  SQUARE CENTIMETERS');
    WRITELN ('FRONT WALL AREA IS : ':33,
            FRONT_WALL_AREA:8:2, '  SQUARE CENTIMETERS');
    WRITELN ('BASE AREA IS : ':33, BASE_AREA:8:2,
             '  SQUARE CENTIMETERS');
    WRITELN ('BOX VOLUME IS : ':33, VOLUME:10:2,
            'CUBIC CENTIMETERS');
   WRITELN (' IT WILL COST : ', COST:7:2,
           'SHEKELS, TO WRAP THE BOX WITH ALUMINUM PAPER')
END.
```

תוכנית 7

הגדרות:
סטיית תקן היא השורש הריבועי של ממוצע ריבועי הסטיות מן הממוצע החשבוני.
שונות היא סטיית התקן בריבוע.

```
PROGRAM MARKS (INPUT, OUTPUT);
VAR
   MARK1, MARK2, MARK3,        { הגדרת הציונים כמספר ממשי כדי }
   AVERAGE,                    { שלא יתקבל OVERFLOW בהעלאה בריבוע }
   STANDARD_DEVIATION : REAL;
BEGIN
   WRITE ('PLEASE ENTER 3 MARKS : ');
   READLN (MARK1, MARK2, MARK3);                    { קרא שלושה ציונים }
   WRITELN;
   AVERAGE := (MARK1 + MARK2 + MARK3) / 3;                  { ממוצע }
   STANDARD_DEVIATION := SQRT ( (3 * (SQR (MARK1)+  { סטיית תקן }
          SQR (MARK2) + SQR (MARK3)) - SQR (3 * AVERAGE)) / 9);
   WRITELN ('THE AVERAGE IS : ',AVERAGE:6:2);
   WRITELN ('DEVIATION OF MARK 1 FROM AVERAGE : ',
             MARK1 - AVERAGE:6:2,'  ABSOLUTE DEVIATION : ',
             ABS (MARK1 - AVERAGE):6:2);            { סטיה מהממוצע }
   WRITELN ('DEVIATION OF MARK 2 FROM AVERAGE : ',
             MARK2 - AVERAGE:6:2,'  ABSOLUTE DEVIATION : ',
             ABS (MARK2 - AVERAGE):6:2);
   WRITELN ('DEVIATION OF MARK 3 FROM AVERAGE : ',
             MARK3 - AVERAGE:6:2,'  ABSOLUTE DEVIATION : ',
             ABS (MARK3 - AVERAGE):6:2);
   WRITELN ('STANDARD DEVIATION : ', STANDARD_DEVIATION:6:2)
END.
```

הערה:

בפתרון של שאלה זו פרקנו את הנוסחה של סטיית התקן לפני החישוב (שורה 11).

תוכנית 8

```
PROGRAM CALCULATE (INPUT, OUTPUT);
CONST
   FACTOR1 = 5;
   FACTOR2 = 10;
VAR
   N : REAL;
BEGIN
   WRITE ('PLEASE ENTER A NUMBER : ');
   READLN (N);                                        { קרא מספר }
   WRITELN;
   WRITELN ('  מספר       ריבוע            שורש        ',
            FACTOR1:2, ' כפול', FACTOR2:2 , ' כפול');
   WRITELN (N-3:8:2, SQR (N-3):12:2, SQRT (N-3):17:2,
            (N-3) * FACTOR1:18:2, (N-3) * FACTOR2:18:2);
   WRITELN (N-2:8:2, SQR (N-2):12:2, SQRT (N-2):17:2,
            (N-2) * FACTOR1:18:2, (N-2) * FACTOR2:18:2);
   WRITELN (N-1:8:2, SQR (N-1):12:2, SQRT (N-1):17:2,
            (N-1) * FACTOR1:18:2, (N-1) * FACTOR2:18:2);
   WRITELN (N+1:8:2, SQR (N+1):12:2, SQRT (N+1):17:2,
            (N+1) * FACTOR1:18:2, (N+1) * FACTOR2:18:2);
   WRITELN (N+2:8:2, SQR (N+2):12:2, SQRT (N+2):17:2,
            (N+2) * FACTOR1:18:2, (N+2) * FACTOR2:18:2)
END.
```

תוכנית 9

```
PROGRAM CALCULATE (INPUT, OUTPUT);
CONST
   PI = 3.141592;
VAR
   RADIUS : REAL;
BEGIN
   WRITE ('PLEASE ENTER CIRCLE RADIUS : ');
   READLN (RADIUS);                                   { קרא רדיוס מעגל }
   WRITELN;
   WRITELN ('THE DIAMETER IS: ':29, 2 * RADIUS:7:2);           { קוטר }
   WRITELN ('THE AREA IS: ':29, PI * SQR (RADIUS):7:2);        { שטח }
   WRITELN ('THE SCOPE IS: ':29, 2 * PI * RADIUS:7:2)          { היקף }
END.
```

תוכנית 10

```
PROGRAM FIBONACHI (INPUT, OUTPUT);
VAR
   FIB_1, FIB_2 : INTEGER;                    { שים לב לערך המכסימלי שיכול }
BEGIN                                         { להיות במשתנים אלה.          }
   WRITE ('PLEASE ENTER TWO FIBONACHI NUMBERS : ');
   READLN (FIB_1, FIB_2);                 { הכנס שני מספרים מסדרת פיבונצ'י }
   WRITELN;
   WRITELN ('THE FOLOWING THREE NUMBERS ARE : ':37, FIB_1 +
            FIB_2:7, FIB_1+ 2 * FIB_2:7, 2 * FIB_1 + 3 * FIB_2:7)
END.
```

הסבר:

חמישה מספרים כלשהם מסדרת פיבונצ'י: a b c d e

נתונים שני מספרים מהסדרה הזו: a b

c=a+b

d=b+c=b+a+b=a+2b

e=c+d=a+b+a+2b=2a+3b

סדרת פיבונצ'י לחמישה מספרים: a b a+b a+2b 2a+3b

תוכנית 11

```
PROGRAM TEMP (INPUT, OUTPUT);
VAR
   FAR_TEMP, CEL_TEMP : REAL;
BEGIN
   WRITE ('ENTER THE TEMPERATURE IN FAHRENHEIT DEGREES : ');
   READLN (FAR_TEMP);                     { קרא טמפרטורה במעלות פרנהייט }
   WRITELN;
   CEL_TEMP := (FAR_TEMP - 32) * 5 / 9;                       { הנוסחה }
   WRITELN ('THE TEMPERATURE IN CELSIUS DEGREES IS : ':49,
            CEL_TEMP:6:2)
END.
```

תוכנית 12

```
PROGRAM EQUATION (INPUT, OUTPUT);
VAR                                           { המשוואה הריבועית היא: }
   A, B, C,                                     { Ax^2 + Bx + C = 0 }
   ROOT, X1, X2 : REAL;
BEGIN
   WRITELN ('ENTER THE EQUATION'S PARAMETERS : ');
```

```
   WRITE ('A = ':33);
   READLN (A);
   WRITE ('B = ':33);
   READLN (B);
   WRITE ('C = ':33);
   READLN (C);
   WRITELN;
   ROOT := SQRT (SQR (B) - 4 * A * C);        {חישוב שורש הדיסקרמיננטה }
   X1 := (-B + ROOT) / (2 * A);
   X2 := (-B - ROOT) / (2 * A);
   WRITELN ('THE ROOTS ARE: X1 = ', X1:7:2, '   X2 = ', X2:7:2)
END.
```

<u>הערה:</u> ראה גם תוכנית 6 בפרק 4.

תוכנית 13

```
PROGRAM TRIANGLE_AREA (INPUT, OUTPUT);
VAR
   A, B, C,                                    { צלעות המשולש }
   S, AREA : REAL;
BEGIN
   WRITE ('ENTER THE TRIANGLE SIZE :   A = ');
   READ (A);
   WRITE ('    B = ');
   READ (B);
   WRITE ('    C = ');
   READLN (C);
   WRITELN;
   S := (A + B + C) / 2;                       { חישוב מחצית ההיקף }
   WRITELN ('THE AREA IS : ',                  { חישוב והדפסה של שטח המשולש }
            SQRT (S* (S-A) * (S-B) * (S-C)):7:2, ' SQR CM.')
END.
```

תוכנית 14

```
PROGRAM BANK (INPUT, OUTPUT);
VAR
   ACOUNT_NUMBER, BALANCE, MONEY : REAL;
BEGIN
   WRITE ('PLEASE ENTER ACOUNT NUMBER : ');
   READLN (ACOUNT_NUMBER);                         { מספר חשבון          }
   WRITE ('  WHAT IS THE LAST BALANCE? ');
   READLN (BALANCE);                               { היתרה האחרונה       }
   WRITE (' WHAT IS THE MONEY MOVEMENT? ');
   READLN (MONEY);                                 { התנועה האחרונה      }
   WRITELN;
   BALANCE := BALANCE + MONEY;                     { המאזן החדש          }
   WRITELN ('ACOUNT NUMBER : ', ACOUNT_NUMBER:7:0,
            '     THE NEW BALANCE IS : ', BALANCE:4:2, ' IS')
END.
```

הסבר:

מספר החשבון מוגדר כמספר ממשי, כדי שלא תהיה גלישה בקריאת מספר הגדול מ-MAXINT. הדפסת מספר חשבון נעשית ללא נקודה עשרונית.
כאשר מושכים כסף, יש לכתוב את הסימן מינוס משמאל למספר.

תוכנית 15

```
PROGRAM CAR (INPUT, OUTPUT);
VAR
   LITER_PRICE, DISTANCE, KM_PER_LITER,
   TOTAL_COST         : REAL;
BEGIN
   WRITE ('ENTER FUEL PRICE FOR ONE LITER : ');
   READLN (LITER_PRICE);                           { מחיר ליטר בנזין }
   WRITE ('WHAT IS THE DISTANCE? ':33);
   READLN (DISTANCE);                              { מרחק }
   WRITE ('CAR EFFICIENCY - KM PER LITER : ':33);
   READLN (KM_PER_LITER);                          { צריכת דלק לקילומטר }
   WRITELN ('TOTAL COST OF FUEL WILL BE ':31,      { חישוב והדפסה }
           LITER_PRICE * DISTANCE / KM_PER_LITER :7:2, ' IS')
END.
```

תוכנית 16

```
PROGRAM WEIGHT (INPUT, OUTPUT);
CONST
   LIB_TO_KG  = 0.45359;
   LIB_TO_GR  = 453.59;
   LIB_TO_OUN = 16;
VAR
   LIBRAS : REAL;
BEGIN
   WRITE ('ENTER THE WEIGHT IN LIBRAS : ');
   READLN (LIBRAS);                                    { משקל בליברות }
   WRITELN ('THE WEIGHT IS EQUAL TO ',
      LIB_TO_KG * LIBRAS:4:2,' KG OR ',LIB_TO_GR * LIBRAS:4:2,
      ' GRAMS OR ',LIB_TO_OUN * LIBRAS:4:2,' OUNCES')
END.
```

תוכנית 17

```
PROGRAM HOUSE (INPUT, OUTPUT);
VAR
   COMMISSION, INTEREST, PRICE, EXPENSES, LEFT : REAL;
BEGIN
   WRITE ('   WHAT IS THE HOUSE PRICE? ');
   READLN (PRICE);                                     { מחיר הבית }
   WRITE ('WHAT IS THE AGENT INTEREST? ');
   READLN (INTEREST);                { העמלה של הסוכן ב-% ממחיר הבית }
   WRITE ('HOW MUCH WERE THE EXPENSES? ');
   READLN (EXPENSES);                                  { הוצאות }
   LEFT :=  PRICE * (1 - INTEREST / 100) - EXPENSES;
   WRITELN('THE OWNER GETS ', LEFT:4:2, ' IS')      { כמה קיבל הבעלים }
END.
```

תוכנית 17 א'

```
PROGRAM HOUSE_A (INPUT, OUTPUT);
VAR
   COMMISSION, INTEREST, PRICE, EXPENSES, LEFT : REAL;
BEGIN
   WRITE ('   WHAT IS THE HOUSE PRICE ? ');
   READLN (PRICE);                                     { מחיר הבית }
   WRITE ('HOW MUCH WERE THE EXPENSES ? ');
   READLN (EXPENSES);                                  {   הוצאות }
   WRITE ('HOW MUCH THE OWNER GETS ? ');
   READLN (LEFT);                         { כמה קיבל לידיו בעל הבית }
```

```
   INTEREST := (PRICE-EXPENSES-LEFT) / PRICE * 100;        { העמלה ב-% }
   WRITE ('THE AGENT INTEREST IS ', INTEREST:4:2, ' PERCENT')
END.
```

תוכנית 17 ב'

```
PROGRAM HOUSE_B (INPUT, OUTPUT);
VAR
   COMMISSION, INTEREST, PRICE, EXPENSES, LEFT : REAL;
BEGIN
   WRITE ('   WHAT IS THE HOUSE PRICE? ');
   READLN (PRICE);                                          { מחיר הבית }
   WRITE ('WHAT IS THE AGENT INTEREST? ');
   READLN (INTEREST);                                   { עמלת הסוכן ב-% }
   WRITE ('   HOW MUCH THE OWNER GETS? ');
   READLN (LEFT);                    { הסכום שקיבל בעל הבית לאחר המכירה }
   EXPENSES := PRICE * (1 - INTEREST / 100) - LEFT;    {חישוב ההוצאות }
   WRITE ('            THE EXPENSES WERE ', EXPENSES:4:2, ' IS')
END.
```

תוכנית 18

```
PROGRAM INVESTMENT (INPUT, OUTPUT);
VAR
   INVEST, INTEREST, SUM : REAL;
   MONTHS         : INTEGER;
BEGIN
   WRITE ('WHAT IS THE INVESTMENT SIZE? ');
   READLN (INVEST);                                     { ההשקעה, הקרן }
   WRITE ('  WHAT IS THE INTEREST RATE? ');
   READLN (INTEREST);                                    { הריבית ב-% }
   WRITE ('        FOR HOW MANY MONTHS? ');
   READLN (MONTHS);                               { מספר חודשי ההשקעה }
   WRITELN ('THE TOTAL SUM IS ':30,                    { הקרן + הריבית }
      INVEST * EXP (LN (1+INTEREST/100) * MONTHS):4:2, ' IS')
END.
```

תוכנית 19

```
PROGRAM FOOD (INPUT, OUTPUT);
CONST
   KG_CHICKEN  = 3.5;                    { מחירי המצרכים השונים }
   KG_MINCE    = 1.7;                    { מחיר לק"ג            }
   KG_LIVER    = 6.8;
   KG_PIE      = 0.6;
   KG_SALAD    = 0.84;
   KG_SHNITZEL = 3.1;
VAR
   CHICKEN, MINCE, LIVER, PIE, SALAD, SHNITZEL, SUM : REAL;
BEGIN
   WRITELN
     ('PLEASE ENTER THE WEIGHT OF THE FOLLOWING GROCERYS : ');
   WRITE ('CHICKEN : ':20);
   READLN (CHICKEN);
   WRITE ('MINCE : ':20);
   READLN (MINCE);
   WRITE ('LIVER : ':20);
   READLN (LIVER);
   WRITE ('PIE : ':20);
   READLN (PIE);
   WRITE ('SALAD : ':20);
   READLN (SALAD);
   WRITE ('SHNITZEL : ':20);
   READLN (SHNITZEL);
   WRITELN ('========================':28);
   SUM := CHICKEN * KG_CHICKEN + MINCE * KG_MINCE +
          LIVER * KG_LIVER +  PIE * KG_PIE + SALAD *
          KG_SALAD + SHNITZEL * KG_SHNITZEL;        { ערך הקניה }
   WRITELN ('SUM : ':20, SUM:4:2);
   WRITELN ('TAX : ':20, SUM * 0.15:4:2);
   WRITELN ('TOTAL : ':20, SUM * 1.15:4:2, 'IS.  THANK YOU ')
END.
```

תוכנית 20

```
PROGRAM TOTO (INPUT,OUTPUT);
VAR
   WINNERS, NUMBER_OF_COPIES : INTEGER;
   COPY_PRICE, WIN           : REAL;
BEGIN
   WRITE ('WHAT IS THE PRICE OF ONE COPY? ');
   READLN (COPY_PRICE);                         { מחיר עותק אחד }
   WRITE ('     HOW MANY COPIES WERE SOLD? ');
   READLN (NUMBER_OF_COPIES);                   { מספר עותקים   }
   WRITE ('      HOW MANY WINNERS WE HAVE? ');
```

```
    READLN (WINNERS);
                                      { כמה אנשים ניחשו נכון }
                   { סכום הזכיה הוא 75% מהכנסות מחולק במספר הזוכים }
    WIN := 0.75 * COPY_PRICE * NUMBER_OF_COPIES / WINNERS;
    WRITELN ('EVERY WINNER WILL GET ':32, WIN:4:2, ' IS')
END.
```

תוכנית 21

```
PROGRAM EQUAL (INPUT, OUTPUT);
VAR
    A, B, C, D, E, F, X, Y, DETERMINANT : REAL;
BEGIN
    WRITELN (
     'THIS PROGRAM SOLVES LINEAR EQUATIONS OF THE FORM :');
    WRITELN ('Ax + By = C':30);
    WRITELN ('Dx + Ey = F':30);
    WRITELN ('PLEASE ENTER YOUR EQUATIONS PARAMETERS :');
    WRITE ('A = ');
    READ (A);
    WRITE ('B = ':10);
    READ (B);
    WRITE ('C = ':10);
    READLN (C);
    WRITE ('D = ');
    READ (D);
    WRITE ('E = ':10);
    READ (E);
    WRITE ('F = ':10);
    READLN (F);
    WRITELN;
    DETERMINANT := A * E - B * D;
    WRITE ('SOLUTION :  X = ', (C * E - B * F) / DETERMINANT:4:2);
    WRITELN ('  Y = ', (A * F - C * D) / DETERMINANT:4:2)
END.
```

<u>הסבר:</u>

פתרון שאלה זו כפוף להנחות שניתנו בשאלה. כלומר: דטרמיננטה שונה מאפס. במלים אחרות, ניקח את מקדמי המשתנים של המשוואות הנתונות, כך:

צריך להיות

```
I.  Ax + By = E         | A B |
II. Cx + Dy = F  ===>>  | C D |  ===>> A*D-B*C <> 0
```

פרק 2
משתנים וביטויים בוליאניים

שאלות

שאלה 1

סדר ההערכה ירשם בסוגריים:

(a) A OR (B AND C) OR D = TRUE
(b) (A AND B) OR (C AND D) = FALSE
(c) (A AND (NOT B)) OR ((NOT A) AND B) = TRUE
(d) A=(B OR C)>=0 פעולה לא-חוקית כי מתבצע
(e) (A = B) OR (C >= D) = TRUE
(f) ביטוי מסוג: משתנה, יחס, משתנה, יחס, משתנה - אינו חוקי
A <> (B AND C) <> D
(g) (A <> B) AND (C <> D) = FALSE

שאלה 2

הפישוט של הביטוי יעשה לפי חוקי דה-מורגן:

* נפעיל את כלל מספר 1 על הסוגריים הפנימיים ((NOT A OR (X<Y)). כאשר (X<Y), ערך הביטוי יהיה שווה ל- NOT(X>=Y).

* לפי חוק מספר 3, שני ה- "NOT" מבטלים זה את זה.

כתוצאה נקבל:

NOT ((NOT A OR (X < Y)) AND (P = Q) ==>> A AND (X >= Y) AND (P = Q)

שאלה 3

```
I DIV J = I/J;
I DIV 2 = 1;
ABS (ABS (I)-4.5) > 0.5
```

שאלה 4

הביטויים שערכם TRUE (קרא משמאל לימין):

```
a) 2*I-1<J ==> 2*4-1<8 ==> 8-1<8 ==> 7<8 = TRUE

b) (I>0) AND (I<=10) ==> (4>0) AND (4<=10) ==> TRUE AND TRUE = TRUE

c) (I>25) OR (I<50) AND (J<50) ==> (4>25) OR ((4<50) AND (8<50))==>
                                 ==>>FALSE OR (TRUE AND TRUE)= TRUE

d) (I<4) OR (J>5) ==> (4<4) OR (8>5) ==> FALSE OR TRUE = TRUE

e) NOT (I>6) ==> NOT (4>6) ==> NOT FALSE = TRUE
```



תוכניות

תוכנית 1

```
(1)  PROGRAM OVED (INPUT, OUTPUT);
(2)  VAR
(3)     AGE, EXPERIENCE, B_A : INTEGER;
(4)     A, B, C              : BOOLEAN;
(5)  BEGIN
(6)     WRITE ('MAHU GIL HAMUAMAD? ');
(7)     READLN (AGE);                                        { גיל המועמד }
(8)     WRITE ('LAMUAMAD YES TOAR B.A (1=YES/0=NO)? ');
(9)     READLN (B_A);                                        { תואר    BA }
(10)    WRITE ('KAMA SHANIM SHEL NISAYON? ');
(11)    READLN (EXPERIENCE);     {                         ניסיון בשנים }
(12)    A := AGE > 21;           {                       A=TRUE if AGE > 21 }
(13)    B := B_A = 1;            {      B=FALSE if B_A not equal to 1 }
(14)    C := EXPERIENCE >= 5;    { C=FALSE if EXPERIENCE less then 5 }
(15)    WRITELN;
(16)    WRITELN ('HAMUAMAD MEMALEH KOL HADRISHOT: ',
(17)             A AND B AND C);
(18)    WRITELN ('HAMUAMAD MEMALEH ALEF VEBET OH GIMEL: ',
(19)             A AND B OR C);
(20)    WRITELN ('HAMUAMAD MEMALEH BET OH GIMEL: ', B OR C);
(21)    WRITELN ('HAMUAMAD LO MEMALEH AF LO AHAT MIN HADRISHOT: ',
(22)             NOT (A OR B OR C))
(23) END.
```

<u>הסבר:</u>

17 תנאים א' ב' ו- ג' מתקיימים.

18 תנאים א' ו-ב' או ג' [(א' ו- ב') או ג'].

20 תנאים ב' או ג'.

21-22 תנאים א' או ב' או ג' אינם מתקיימים.

תוכנית 2

```
PROGRAM NUMBERS (INPUT, OUTPUT);
VAR
   A1, A2, B1, B2 : INTEGER;
BEGIN
   WRITE ('PLEASE ENTER THE FIRST PAIR OF INTEGERS : ');
   READLN (A1, A2);                               { זוג מספרים ראשון }
   WRITE ('ENTER THE SECOND PAIR : ');
   READLN (B1, B2);                               { זוג מספרים שני }
   WRITELN;
   WRITELN ('THE FIRST CONDITION IS  ', (A1=B1) AND (A2=B2));
   WRITELN ('THE SECOND CONDITION IS ', (A1=B1) OR (A2=B2));
   WRITELN ('THE THIRD CONDITION IS  ', (A1=B2) OR (A2=B1))
END.
```

תוכנית 3

```
PROGRAM STUDENTS (INPUT, OUTPUT);
VAR
   STUDENT_NUMBER, SEX, POINTS : INTEGER;
   HEIGHT, WEIGHT              : REAL;
BEGIN
   WRITE ('ENTER PLEASE STUDENT NUMBER : ');
   READLN (STUDENT_NUMBER);                    { מספר סטודנט }
   WRITE ('ENTER SEX (MALE=1/FEMALE=0)? ');
   READLN (SEX);                               { מין: נקבה=1, זכר=0 }
   WRITE ('HOW MANY POINTS HE GOT? ');
   READLN (POINTS);                            { מספר נקודות }
   WRITE (' WHAT IS THE STUDENT HEIGHT? ');
   READLN (HEIGHT);                            { גובה הסטודנט/ית }
   WRITE (' WHAT IS THE STUDENT WEIGHT? ');
   READLN (WEIGHT);                            { משקל הסטודנט/ית }
   WRITELN;
   WRITELN ((SEX = 1) AND (POINTS >= 85) AND (HEIGHT > 1.8)
            AND (WEIGHT > 70), ' מתאים להיות איש ביטחון: ',
            STUDENT_NUMBER:4, ' סטודנט מספר ')
END.
```

<u>הסבר:</u>

בהוראת WRITELN (האחרונה בתוכנית זו) כוללים גם את בחינת התנאי וגם הדפסה של תוצאות בחינה זו ושל נתונים אחרים.
בחינת התנאי שבראש ההוראה תפיק את המלים TRUE או FALSE בהתאמה, אשר תודפס במקומה במשפט.

תוכנית 4

```
PROGRAM BOOKS (INPUT, OUTPUT);
VAR
   PAY_FOR_YEAR   : REAL;
   PAY_ON_TIME, YEARS : INTEGER;
   A, B, C    : BOOLEAN;
BEGIN
   WRITE ('PLEASE ENTER THE BOOKS VALUE FOR A YEAR : ');
   READLN (PAY_FOR_YEAR);      { ערך הסחורה שקנה הלקוח במשך השנה }
   WRITE ('DID HE PAY ALL BILLS ON TIME (1=YES/0=NO)? ');
   READLN (PAY_ON_TIME);
   WRITE ('WHAT IS THE SERVICE TIME FOR THIS CLIENT? ');
   READLN (YEARS);
   WRITELN;
   A := PAY_FOR_YEAR > 8000;
   B := PAY_ON_TIME = 1;
   C := YEARS > 5;
   WRITELN (A AND C OR B, 'תן ללקוח טיפול מיוחד: ')
END.
```

<u>הסבר:</u>

ארבעת התנאים מזכים את הלקוח בטיפול מיוחד. אפשר לצמצם אותם בדרך זו:

$$AB+AC+BC+B=AC+B(1+A+C)=AC+B$$

<u>הערה:</u> הסימן '+' הינו סימן בוליאני המבטא פעולת OR.

פרק 3
לולאות

שאלות

שאלה 1

ערכו של S:

לאחר לולאה ראשונה נקבל: S=99.
לאחר לולאה שניה נקבל: S=-1, וזהו גם ערכו של S בסוף התוכנית.

שאלה 2

הרצה "על יבש" של תוכנית זו:

פלט	D	C	B	A	מספר המעבר
	17	0	1	0	ערך התחלתי
0 1	1	1	1	1	1
1 2	2	2	1	3	2
2 3	3	3	1	6	3
3 2	2	4	1	8	4
4 4	4	5	1	12	5
5 5	5	6	1	17	6
6 9	9	7	1	26	7
7					יציאה מהלולאה

הפלט יראה כך: 7 9 6 5 5 4 4 2 3 3 2 2 1 1 0

שאלה 3

בהרצה של התוכנית בשלבים נקבל את התוצאות הבאות:

פלט	D	C	B	A	E	מספר מעבר
	-	0	1	-1	0	ערך התחלתי
1	1	1	0		-1	1
1	1	1	1		-2	2
2	2	2	1		-3	3
3	3	3	2		-4	4
5	5	5	3		-5	5
8	8	8	5		-6	6
13	13	13	8		-7	7
21	21	21	13		-8	8
-8						

הפלט יראה כך: 1 1 2 3 5 8 13 21 -8

שאלה 4

ערכו של MISHTANEH בסוף כל קטע:

א. 10
ב. 515
ג. -3
ד. -2

תוכניות

תוכנית 1

```
PROGRAM TABLE (INPUT, OUTPUT);
VAR
   N, INDEX, SUM : INTEGER;
BEGIN
   SUM := 0;                                              { קביעה של ערך התחלתי }
   WRITELN ('      N        SQR(N)      SQRT(N)          SUM');
   WRITELN ('   ==========================================');
   FOR INDEX := 1 TO 25 DO                                 { לולאה של 25 צעדים }
    BEGIN
      SUM := SUM + INDEX;                          { סיכום בכל שלב מ-1 עד IDNEX }
      WRITELN (INDEX:6, SQR (INDEX):12, SQRT (INDEX):12:3, SUM:12)
    END
END.
```

תוכנית 2

```
PROGRAM FACTORIAL (INPUT, OUTPUT);
VAR
   N, INDEX : INTEGER;
   FACT     : REAL;           { המשתנה מוגדר כך, כדי שלא יהיה OVERFLOW }
BEGIN
   WRITE ('N =? ');
   READLN (N);
   FACT := N;                                           { ערך התחלתי }
   FOR INDEX := N-1 DOWNTO 1 DO FACT := FACT * INDEX;
   WRITELN ('N! = ', FACT:1:0)
END.
```

<u>הסבר:</u>

הערך של "N עצרת" מחושב כך: N! = N * (N-1) * (N-2) *...* 1

שים לב ש: 8! = 40320 לכן FACT מוגדר כ-REAL, ולא INTEGER.

תוכנית 3

```
(1)  PROGRAM BANK (INPUT, OUTPUT);
(2)  VAR
(3)     MONTH, MONEY_IN, MONEY_OUT : INTEGER;
(4)     SUM, INTEREST, TOTAL       : REAL;
(5)  BEGIN
(6)     MONTH     := 0;                          { ערכים התחלתיים }
(7)     MONEY_IN  := 200;
(8)     MONEY_OUT := 100 + 350;
(9)     TOTAL     := 2500;
(10)    WRITELN ('יתרה בסוף חודש        יתרה בסוף חודש');
(11)    WRITELN (
(12)    'חודש   הכנסה   הוצאות   לפני ריבית   ריבית   אחרי ריבית ');
(13)    WRITELN (
(14)    '------------------------------------------------------------');
(15)    WHILE  TOTAL > 0  DO
(16)     BEGIN
(17)       MONTH := MONTH + 1;                     { מונה החודשים }
(18)       SUM := TOTAL + MONEY_IN - MONEY_OUT;
(19)       INTEREST := SUM * 0.02;                 { חישוב הריבית }
(20)       TOTAL := SUM + INTEREST;
(21)       WRITELN (TOTAL:10:2, INTEREST:10:2, SUM:10:2,
(22)          MONEY_OUT:10, MONEY_IN:8, MONTH:8)
(23)     END
(24) END.
```

תוכנית 4

לפתרון בעיה זו יש להחליף בתשובה לתוכנית 3 את שורות 7 ו-8, בפקודת READ מתאימה.

תוכנית 7

```
PROGRAM PRIME (INPUT, OUTPUT);
VAR
   PRIME    : BOOLEAN;
   INDEX, N : INTEGER;
BEGIN
   WRITE ('N = ');
   READLN (N);
   INDEX := 2;                               { המחלק הראשון יהיה 2 }
   WHILE (INDEX < SQRT (N)) AND (N MOD INDEX > 0) DO
      INDEX := INDEX + 1;
   PRIME := NOT (N MOD INDEX = 0);
   WRITELN (PRIME, ' הוא ראשוני ', N:1, ' המספר')
END.
```

הסבר:

כדי למצוא את מחלקיו של מספר, מספיק לבדוק עד שורש המספר, ואין צורך לבדוק את כל האפשרויות שאחריו.

להלן מספר דוגמאות שיבהירו זאת:

דוגמה 1 — מספר 12: sqrt(12) = 3.4641

מחלק	מחולק	
2	6	
3	4	—
4	3	
6	2	

דוגמה 2 — מספר 36: sqrt(36) = 6

מחלק	מחולק	
2	18	
3	12	
4	9	
6	6	—
9	4	
12	3	
18	2	

תוכל להבחין בכך שהטור של המחלק זהה לטור המחולק, אך הפוך לו.

הערה: אין צורך לבדוק את כל המספרים הזוגיים. מספיק לבדוק את 2, כי N מתחלק במספר זוגי כלשהו רק אם הוא מתחלק גם ב-2.
פתרון לבעיה בדרך זו ניתן למצוא בתוכנית 12 שבפרק 4 בספר זה.

9 שני תנאים נבדקים בפקודה זו:
א. אם index קטן משורש N.
ב. אם N לא מתחלק ב-index ללא שארית.
הלולאה נמשכת כל זמן ששני התנאים מתקיימים (and).

11 א. במקרה שהיציאה היתה רק בגלל אי-קיום התנאי הראשון - PRIME יקבל TRUE. כלומר - לא נמצא מספר שמחלק את N ללא שארית.
ב. כאשר היציאה מהלולאה היתה בגלל שהתנאי השני לא התקיים, המשמעות היא ש-N התחלק ב-INDEX ללא שארית, ואז PRIME יקבל FALSE.

אם שני התנאים לא התקיימו - PRIME יקבל FALSE בדומה למקרה השני.

תוכנית 8

```
PROGRAM ROOT (INPUT, OUTPUT);
VAR
   X, D, A : REAL;
BEGIN
   WRITE ('X = ');
   READLN (X);
   D := 0.001;                                { הדיוק }
   A := X/2;                                  { ערך התחלתי }
   WHILE  ABS (X - SQR (A)) >= D * X  DO
      A := (A + X / A) / 2;
   WRITELN ('A = ', A:6:4, '      DX = ', D * X :6:4)
END.
```

<u>הסבר:</u>

9-10 כל זמן שהערך המוחלט של השגיאה גדול או שווה למכפלה D*X,
מוצאים A חדש לפי הנוסחה.
היציאה מהלולאה תתבצע כאשר השגיאה קטנה מהמכפלה D*X.

תוכנית 9

```
PROGRAM ROOT (INPUT, OUTPUT);
VAR
   INDEX, COUNTER : INTEGER;
   X, D, A        : REAL;                                { D = Delta }
BEGIN
   WRITE ('X = ');
   READLN (X);
   WRITELN ('    A          D          DX         STEPS');
   WRITELN ('  ---------------------------------------');
   FOR  INDEX := 1  TO  100 DO
    BEGIN
      COUNTER := 0;                          { מונה מספר הצעדים }
      D := 0.001 * INDEX;
      A := X / 2;                            { ערך התחלתי }
      WHILE  ABS (X - SQR (A)) >= D * X  DO
       BEGIN
         A := (A + X / A) / 2;
         COUNTER := COUNTER + 1
       END; { while }
      WRITELN (A:10:4, D:10:4, D * X:10:4, COUNTER:10)
    END    { for }
END.
```

<u>הסבר:</u>

מציאת השורש בשאלה זו מתבצעת בדומה לפתרון בתוכנית 8. השוני הוא בכך, שכאן נעשה שימוש בלולאה, והערך של D משתנה מ-0.001 עד 0.1. בכל שלב של כניסה מחודשת ללולאה מתאפס המונה וניתן ערך התחלתי ל-A.

<u>הערה:</u>
הערך ההתחלתי (שורה 8 בתוכנית 8 ושורה 14 בתוכנית 9) נקבע כ- X/2. נסה להריץ תוכניות אלו עם ערכים התחלתיים שונים ובחן כיצד מגיעים לפתרון (קריטריון הבחינה - מספר הצעדים).

תוכנית 10

```
(1)  PROGRAM HEAP (INPUT, OUTPUT);
(2)  VAR
(3)     STEP         : INTEGER;
(4)     LENGTH, WIDTH, HEIGHT, TEMP : REAL;
(5)  BEGIN
(6)     STEP   := 0;
(7)     LENGTH := 2;
(8)     WIDTH  := 1.5;
(9)     HEIGHT := 0.05;
(10)    WRITELN ('שלב      רוחב הדיקט      אורך הדיקט      גובה הדיקט ');
(11)    WRITELN ('-----------------------------------------------------');
(12)    WRITELN (HEIGHT:10:2, LENGTH:14:4, WIDTH:14:4, STEP:10);
(13)    REPEAT
(14)       LENGTH := LENGTH / 2;
(15)       HEIGHT := HEIGHT * 2;
(16)       WHILE  LENGTH < WIDTH DO
(17)        BEGIN
(18)          TEMP   := WIDTH;      { החלפת ערכים בין WIDTH ל- LENGTH }
(19)          WIDTH  := LENGTH;     {         בעזרת משתנה עזר בשם TEMP. }
(20)          LENGTH := TEMP
(21)        END;
(22)       STEP := STEP + 1;
(23)       WRITELN (HEIGHT:10:2, LENGTH:14:4, WIDTH:14:4, STEP:10)
(24)    UNTIL  HEIGHT >= 10
(25) END.
```

<u>הסבר:</u>
שורה 12 - הדפסת ערכים התחלתיים.
13-24 - לולאה המתבצעת עד שגובה הערימה גדול או שווה ל- 10 מטר.
14 - אורך המשטח מחולק ב- 2.
15 - גובה המשטח מוכפל ב- 2.
16-21 - כאשר ישנו מצב בו האורך קטן מהרוחב, יש להחליף ביניהם.
22 - מונה השלבים.
23 - הדפסת ערכים.

הערה: ניתן להחליף את שורות 7 עד 9 בפקודת READ מתאימה, כדי להריץ את התוכנית עם ערכים אחרים.

תוכנית 11

```
PROGRAM BIGEST_INTEGER (INPUT, OUTPUT);
VAR
   N : REAL;
BEGIN
   N := 142555.0;                    { N גדול מ-MAXINT ולכן REAL }
   WRITELN (TRUNC (SQRT (N)):4, ' הוא ',
      N:1:0, 'המספר השלם הגדול ביותר שהריבוע שלו קטן מ-')
END.
```

<u>הסבר:</u>

יש להשתמש בהגדרת REAL למשתנה N כדי למנוע INTEGER CONSTANT ERROR.

המספר המבוקש מתקבל על ידי הוצאת שורש ריבועי למספר הנתון (ע"י SQRT) והשמטת השבר שאחרי הנקודה (ע"י TRUNC).

תוכנית 11 א'

דרך נוספת לפתרון בעיה 11, תוך שימוש בלולאות:

```
PROGRAM THE_BIGEST_INTEGER (INPUT, OUTPUT);
VAR
   N, INDEX : REAL;
BEGIN
   N := 142555.0;
   INDEX := 1;                                    { ערך התחלתי }
   WHILE  SQR (INDEX) < N DO INDEX := INDEX + 1;
   WRITELN (INDEX - 1 :1:0, ' הוא ',
      N:1:0, 'המספר השלם הגדול ביותר שהריבוע שלו קטן מ-')
END.
```

<u>הסבר:</u>

כל זמן שריבוע ה-INDEX קטן מ-N, נגדיל אותו באחד (שורה 7). התוצאה היא לבסוף INDEX-1 כי היציאה מהלולאה (שורה 7) מתרחשת רק כשהתנאי לא מתקיים, כלומר: SQR (INDEX) < N.

יש לשים לב לכך ש-INDEX מוגדר כ-REAL כדי שלא תהיה גלישה.

<u>הערה:</u> יתכן שבמהדרים מסוימים הפעולה SQR(IDNEX) תגרום לגלישה, אם IDNEX מוגדר כשלם.

תוכנית 12

```
(1)  PROGRAM POWER (INPUT, OUTPUT);
(2)  VAR
(3)     A, B, N, POWER : INTEGER;
(4)  BEGIN
(5)     WRITE ('A = ');
(6)     READLN (A);
(7)     WRITE ('B = ');
(8)     READLN (B);
(9)     N     := 0;            { ערכים התחלתיים }
(10)    POWER := A;
(11)    WHILE  POWER < B  DO
(12)     BEGIN
(13)       POWER := POWER * A;
(14)       N := N + 1
(15)     END;
(16)    WRITELN (B:1, ' קטן מ- ', N:1, ' בחזקת ',
(17)       A:1, ' הוא השלם הגדול ביותר המקיים ', N:1)
(18) END.
```

<u>הסבר:</u>

יש לשים לב שבלולאה זו היציאה נעשית רק לאחר ש-A^N גדול מ-B. אמנם, ל-N מוסיפים עוד 1 במעבר האחרון בלולאה, אך כיון שערכו ההתחלתי (שורה 9) הוא אפס (0), הרי ערכו בגמר הלולאה יהיה התשובה הנכונה. במקרה של A > B אזי התשובה תהיה N = 0.

תוכנית 13 א'

```
PROGRAM TOOR_A (INPUT, OUTPUT);
VAR
   INDEX, N, SUM : INTEGER;     { MAXINT !!! }
BEGIN
   WRITE ('N = ');
   READLN (N);
   SUM := 0;
   FOR  INDEX := 1  TO  N  DO                    { סיכום של N האיברים בטור }
      SUM := SUM + 2 * INDEX - 1;
   WRITELN ('THE SUM IS ', SUM:1)
END.
```

תוכנית 13 ב'

```
PROGRAM TOOR_B (INPUT, OUTPUT);
VAR
   INDEX, N       : INTEGER;
   X, POWER, SUM : REAL;
BEGIN
   WRITE ('X = ');
   READLN (X);
   WRITE ('N = ');
   READLN (N);
   SUM := 0;
   POWER := 1;
   FOR  INDEX := 1  TO  N  DO
    BEGIN
      SUM := SUM + INDEX * POWER;
      POWER := POWER * X
    END;
   WRITELN ('THE SUM IS ', SUM:4:2)
END.
```

תוכנית 13 ג'

```
PROGRAM TOOR_C (INPUT, OUTPUT);
VAR
   INDEX, N, SIGN : INTEGER;
   X, POWER, SUM  : REAL;
BEGIN
   WRITE ('X = ');
   READLN (X);
   WRITE ('N = ');
   READLN (N);
   SUM := 0;
   POWER := X;
   SIGN := 1;
   FOR  INDEX := 1  TO  N  DO
    BEGIN
      SUM := SUM + SIGN * POWER / (2 * INDEX - 1);
      POWER := POWER * SQR (X);
      SIGN := -1 * SIGN
    END;
   WRITELN ('THE SUM IS ', SUM:4:2)
END.
```

תוכנית 14

```
PROGRAM MONTH_PAY (INPUT, OUTPUT);
VAR
   N        : INTEGER;
   P, I, R, POWER : REAL;
BEGIN
   WRITE ('P = ');
   READLN (P);
   WRITE ('I = ');
   READLN (I);
   WRITE ('N = ');
   READLN (N);
   POWER := EXP ( N * LN (1 + I));
   R := P * I * POWER / (POWER - 1);
   WRITELN (R:4:2, ' התשלום החודשי הוא')
END.
```

הערה:

הסבר על הפונקציה EXP תמצא בע"מ 28 (פרק 1) בספר פסקל.

תוכנית 15

```
PROGRAM LEFT_DIGITS (INPUT, OUTPUT);
VAR
   X      : REAL;
   DIGITS : INTEGER;
BEGIN
   WRITE ('X = ');
   READLN (X);
   DIGITS := 0;
   WHILE  ABS (X) >= 1 DO        {כדי שהתוכנית תפעל עם מספרים שליליים}
    BEGIN
      DIGITS := DIGITS + 1;
      X := X / 10
    END;
   WRITELN ('ספרות משמאל לנקודה ', DIGITS:1, ' במספר זה יש')
END.
```

הסבר:

מספר הספרות משמאל לנקודה העשרונית שווה למספר הפעמים שמנת החלוקה של המספר ב-10 גדולה או שווה ל-1.

פרק 4
הסתעפויות

שאלות

שאלה 1

```
IF ((X < 0) AND (Y < 0)) OR (X = 5) THEN
   A := X + Y + Z;
```

שאלה 2

א. A = 10
ב. A נשאר אותו ערך שהיה בתחילה.

שאלה 3

א. התוכנית מוצאת אם מספר מופיע ברשימת מספרים. אם הוא נמצא ברשימה היא מוצאת את מיקומו.
ב. הפלט הוא המספר המופיע במקום 5 ברשימה.

שאלה 4

א. התוכנית מוצאת את היום הגשום הראשון בחודש, ואם אין יום כזה באותו חודש - היא מודיעה שלא היה אף לא יום גשום אחד במשך החודש.

ב1. משפט ה- WHILE יראה כך:

```
WHILE  ((GASHUM=0) OR (GAVOHA <= 18)) AND
                  (YOM < YAMIM_BACHODESH) DO
IF  (GASHUM > 0) AND (GAVOHA > 18)  THEN
     WRITELN ('היה גשום ומידת החום בו היתה מעל 18 מעלות ',
             YOM:1, ' יום')
```

```
    ELSE
       WRITELN ('במשך החודש שמידת החום בו היתה מעל 18 מעלות ',
                'לא היה אף לא יום גשום אחד ')
```

ב2.

```
WHILE ((GASHUM=0) OR (NAMUCH >= 7)) AND
                (YOM < YAMIM_BACHODESH) DO
  IF  (GASHUM > 0) AND (NAMUCH < 7)  THEN
     WRITELN ('היה גשום ומידת החום בו היתה למטה מ-7 מעלות ',
             YOM:1, ' ביום')
  ELSE
     WRITELN ('במשך החודש שמידת החום בו היתה למטה מ-7 מעלות ',
              'לא היה אף לא יום גשום אחד ')
```

ב3.

```
PROGRAM GESHEM (INPUT, OUTPUT);
VAR
   GASHUM_1, GASHUM_2, GASHUM_3, INDEX, DAYS,
   GAVOHA, NAMUCH, YOM, YAMIM_BACHODESH     : INTEGER;
BEGIN
   READLN (YAMIM_BACHODESH);
   DAYS := 0;
   INDEX := 3;      { שלשת הימים הראשונים }
   READLN (GASHUM_1, GAVOHA, NAMUCH);
   READLN (GASHUM_2, GAVOHA, NAMUCH);
   YOM := 2;
   WHILE  (YOM < YAMIM_BACHODESH) AND (DAYS < 3)  DO
    BEGIN
      READLN (GASHUM_3, GAVOHA, NAMUCH);
      YOM := YOM + 1;
      IF  DAYS < GASHUM_1 + GASHUM_2 + GASHUM_3  THEN
       BEGIN
         DAYS := GASHUM_1 + GASHUM_2 + GASHUM_3;
         INDEX := YOM
       END;  { if }
      GASHUM_1 := GASHUM_2;
      GASHUM_2 := GASHUM_3
    END;      { while }
   WRITELN ('היו הגשומים ביותר במשך החודש', INDEX:3,
                              INDEX - 1:3, INDEX - 2:3, ' הימים')
END.
```

<u>הערה:</u>

היות ואין התייחסות לכמות הגשם שירדה, אלא אם ירד גשם או לא, יש לשמור על היום האחרון הגשום בסדרה האחרונה של ימי גשם רצופים.

הסבר:

שני הימים האחרונים נשמרים בשורות 21 ו-22.
16-20 המשתנה DAYS מקבל את המספר המכסימלי של ימים גשומים במשך שלושה ימים רצופים כלשהם במשך בחודש. כאשר DAYS מקבל ערך חדש, כלומר נמצאו יותר ימים גשומים מבין 3 ימים רצופים, מתעדכן משתנה INDEX ומצביע על היום האחרון בשלישיה זו.

הערה:

בפתרון שאלה זו ובתרגילים הבאים בצענו את פעולות הקלט והפלט הנדרשות לפי תנאי הבעיה בלבד. בעבודה מעשית יש לתת הנחיות לשם קבלת הקלט הנדרש ולתת כותרות והסברים מפורטים יותר בפלט. התלמיד יוכל לעשות זאת בעצמו.

תוכניות

תוכנית 1

```
PROGRAM B_DIV_A (INPUT, OUTPUT);
VAR
   A, B              : INTEGER;
   B_MOD_A_IS_ZERO : BOOLEAN;
BEGIN
   WRITELN ('הכנס 2 מספרים שלמים');
   WRITE ('A = ');
   READLN (A);
   WRITE ('B = ');
   READLN (B);
   B_MOD_A_IS_ZERO := B MOD A = 0;
   WRITELN (B_MOD_A_IS_ZERO, ' B מתחלק ב-A ללא שארית')
END.
```

תוכנית 2

```
PROGRAM MARKET (INPUT, OUTPUT);
VAR
   SUM, COST, REDUCTION_POINT : REAL;
BEGIN
   READLN (REDUCTION_POINT);
   SUM := 0;
   REPEAT
      READLN (COST);
```

```
(9)         SUM := SUM + COST
(10)     UNTIL  COST = 0;
(11)     WRITELN;
(12)     IF  SUM < REDUCTION_POINT THEN
(13)        WRITELN ('הקונה לא מקבל הנחה ')
(14)     ELSE
(15)         BEGIN
(16)           WRITELN ('הקונה מקבל הנחה של 5%');
(17)           SUM := SUM * 0.95
(18)         END;
(19)     WRITELN (SUM:4:2, 'חשבון סופי לתשלום: ')
(20) END.
```

הסבר:

נקבע שהערך שמסמן את סוף סדרת המספרים בקלט יהיה אפס (0).

בלולאה 7-10 ה-COST האחרון (שמסמל את סוף הרשימה) מחובר לסכום, אך מכיון שערכו 0 הוא אינו משנה את התוצאה.

תוכנית 3

```
(1)  PROGRAM TRIANGLE (INPUT, OUTPUT);
(2)  VAR
(3)     A, B, C : REAL;
(4)  BEGIN
(5)     WRITELN ('הכנס 3 מספרים המייצגים את אורכי הצלעות של משולש ');
(6)     READLN (A, B, C);
(7)     IF (A + B < C) OR (A + C < B) OR (B + C < A) THEN
(8)        WRITELN ('הצלעות אינן יכולות להרכיב משולש ')
(9)     ELSE
(10)       IF (A = B) AND (B = C) THEN
(11)          WRITELN ('משולש שווה צלעות ')
(12)       ELSE
(13)          IF (A = B) OR (B = C) OR (A = C) THEN
(14)             WRITELN ('משולש שווה שוקיים ')
(15)          ELSE
(16)             WRITELN ('משולש שונה צלעות ')
(17) END.
```

הסבר:

שורה 7 מציגה בדיקה של הקלט על פי המשפט: במשולש, סכום של שתי צלעות גדול מהצלע השלישית.

תוכנית 4

```
PROGRAM COLORS (INPUT, OUTPUT);
LABEL
   11, 12, 13, 21, 22, 23, 31, 32, 33, 41, 42, 43, 99;
VAR
   HAIR, EYE, COUNTER,
   BLACK_BLUE, BLACK_BROWN, BLACK_GREEN,
   BROWN_BLUE, BROWN_BROWN, BROWN_GREEN,
   RED_BLUE,   RED_BROWN,   RED_GREEN,
   BLOND_BLUE, BLOND_BROWN, BLOND_GREEN  : INTEGER;
BEGIN
   COUNTER     := 0;
   BLACK_BLUE  := 0;
   BLACK_BROWN := 0;
   BLACK_GREEN := 0;
   BROWN_BLUE  := 0;
   BROWN_BROWN := 0;
   BROWN_GREEN := 0;
   RED_BLUE    := 0;
   RED_BROWN   := 0;
   RED_GREEN   := 0;
   BLOND_BLUE  := 0;
   BLOND_BROWN := 0;
   BLOND_GREEN := 0;
   WHILE NOT EOF DO
    BEGIN
      READLN (HAIR, EYE);
      COUNTER := COUNTER + 1;
      CASE HAIR OF
         1: IF EYE = 1 THEN
               GOTO 11
            ELSE
               IF EYE = 2 THEN
                  GOTO 12
               ELSE  { EYE = 3 }
                  GOTO 13;
         2: IF EYE = 1 THEN
               GOTO 21
            ELSE
               IF EYE = 2 THEN
                  GOTO 22
               ELSE  { EYE = 3 }
                  GOTO 23;
         3: IF EYE = 1 THEN
               GOTO 31
            ELSE
               IF EYE = 2 THEN
                  GOTO 32
               ELSE  { EYE = 3 }
                  GOTO 33;
```

```
             4: IF EYE = 1 THEN
                   GOTO 41
                ELSE
                   IF EYE = 2 THEN
                      GOTO 42
                   ELSE  { EYE = 3 }
                      GOTO 43
       END;
 11:   BLACK_BLUE := BLACK_BLUE + 1;
       GOTO 99;
 12:   BLACK_BROWN := BLACK_BROWN + 1;
       GOTO 99;
 13:   BLACK_GREEN := BLACK_GREEN + 1;
       GOTO 99;
 21:   BROWN_BLUE := BROWN_BLUE + 1;
       GOTO 99;
 22:   BROWN_BROWN := BROWN_BROWN + 1;
       GOTO 99;
 23:   BROWN_GREEN := BROWN_GREEN + 1;
       GOTO 99;
 31:   RED_BLUE := RED_BLUE + 1;
       GOTO 99;
 32:   RED_BROWN := RED_BROWN + 1;
       GOTO 99;
 33:   RED_GREEN := RED_GREEN + 1;
       GOTO 99;
 41:   BLOND_BLUE := BLOND_BLUE + 1;
       GOTO 99;
 42:   BLOND_BROWN := BLOND_BROWN + 1;
       GOTO 99;
 43:   BLOND_GREEN := BLOND_GREEN + 1
 99: END;  { while not eof }
    WRITELN ('צבע שיער   צבע עיניים  אחוז מהאוכלוסיה');
    WRITELN ('-----------------------------------------');
    WRITELN (BLACK_BLUE  / COUNTER*100:12:2,
       'שחור          כחול          ');
    WRITELN (BLACK_BROWN / COUNTER*100:12:2,
       'שחור          חום           ');
    WRITELN (BLACK_GREEN / COUNTER*100:12:2,
       'שחור          ירוק          ');
    WRITELN (BROWN_BLUE  / COUNTER*100:12:2,
       'חום           כחול          ');
    WRITELN (BROWN_BROWN / COUNTER*100:12:2,
       'חום           חום           ');
    WRITELN (BROWN_GREEN / COUNTER*100:12:2,
       'חום           ירוק          ');
    WRITELN (RED_BLUE    / COUNTER*100:12:2,
       'ג׳ינג׳י        כחול          ');
    WRITELN (RED_BROWN   / COUNTER*100:12:2,
       'ג׳ינג׳י        חום           ');
    WRITELN (RED_GREEN   / COUNTER*100:12:2,
```

```
          '           ירוק        ג'ינג'י');
(93)    WRITELN (BLOND_BLUE  / COUNTER*100:12:2,
          '           כחול        בלונדיני');
(94)    WRITELN (BLOND_BROWN / COUNTER*100:12:2,
          '           חום         בלונדיני');
(95)    WRITELN (BLOND_GREEN / COUNTER*100:12:2,
          '           ירוק        בלונדיני');
(96)    WRITELN;
(97)    WRITELN (COUNTER:20, ' גודל האוכלוסיה הוא:');
(98) END.
```

הסבר:

11-23 קביעה של ערכים התחלתיים לכל המונים (אפס). לכל אחת מהאפשרויות קיים משתנה, אשר מקבל ערך התחלתי אפס (0).

28-57 פקודת CASE לפי ערך HAIR - צבע השיער. בתוכה ישנה פקודת CASE נוספת, "מלאכותית", בעזרת פקודת GOTO, אשר מעבירה את התוכנית לקטעים שבהם נוסף 1 למשתנה הרלוונטי לאפשרות שנבחרה.

תיאור גרפי של פקודת ה- CASE נראה כך:



תוכנית 5

```
PROGRAM DIGITS (INPUT, OUTPUT);
VAR
   A, B  : INTEGER;
   DIGIT : BOOLEAN;
BEGIN
   READLN (A, B);
   DIGIT := FALSE;
   WHILE  (B > 0) AND NOT DIGIT  DO
    BEGIN
      DIGIT := B MOD 10 = A;
      B := B DIV 10
    END;
   WRITELN (DIGIT, 'הספרה A מופיעה ב-B: ')
END.
```

תוכנית 6

```
PROGRAM SQURE_EQUATION (INPUT, OUTPUT);
VAR
   A, B, C, D, X1, X2, I : REAL;
BEGIN
   WRITELN ('הכנס את שלושת מקדמי המשוואה ');
   READLN (A, B, C);
   D := SQR (B) - 4 * A * C;
   IF  D = 0  THEN
    BEGIN
      X1 := - B / (2 * A) ;
      WRITELN (X1:4:2, 'יש רק שורש אחד:  ')
    END
   ELSE
      IF  D > 0 THEN
       BEGIN
         X1 := (-B + SQRT (D) ) / (2 * A);
         X2 := (-B - SQRT (D) ) / (2 * A);
         WRITELN (X1:4:2, ' ,  ', X2:4:2,
                  'יש שני שורשים ממשיים:  ')
       END
      ELSE      { D < 0 }
       BEGIN
         X1 := -B / (2 * A);
         I  := SQRT (ABS (D) );
         WRITELN (X1:4:2, '+i', I:4:2, ' ,  ', X1:4:2, '-i',
                  I:4:2, 'יש שני שורשים לא ממשיים:  ')
       END
END.
```

הסבר:

כאשר D < 0 ישנם שורשים לא ממשיים (מרוכבים).

תוכנית 7

```
PROGRAM SORT (INPUT, OUTPUT);
VAR
   A, B, C, TEMP : REAL;
BEGIN
   WRITELN ('הכנס 3 מספרים ');
   READLN (A, B, C);
   WHILE  (B < A) OR (C < B)  DO
    BEGIN
      IF  A > B  THEN
       BEGIN
         TEMP := A;      { החלפה בין A ל- B בעזרת משתנה עזר TEMP }
         A    := B;
         B    := TEMP
       END;
      IF  C < B  THEN
       BEGIN
         TEMP := B;
         B    := C;
         C    := TEMP
       END
    END;
   WRITELN (A:10:2, B:10:2, C:10:2)
END.
```

תוכנית 8

```
PROGRAM MARKS (INPUT, OUTPUT);
VAR
   HISTORY, MATHEMATICS,
   H_COUNTER, M_COUNTER   : INTEGER;
BEGIN
   H_COUNTER := 0;
   M_COUNTER := 0;
   READLN (HISTORY, MATHEMATICS);
   WHILE  HISTORY <> -1  DO           { 1- מסמן את סוף סדרת הקלט }
    BEGIN
      IF HISTORY > MATHEMATICS THEN
         H_COUNTER := H_COUNTER + 1
      ELSE
```

```
          IF HISTORY < MATHEMATICS THEN
             M_COUNTER := M_COUNTER + 1;
       { השיויון HISTORY = MATHEMATICS לא נספר }
       READLN (HISTORY, MATHEMATICS)
     END;
   WRITELN (' תלמידים קיבלו ציון יותר גבוה במתמטיקה. ',
                M_COUNTER:1);
   WRITELN (' תלמידים קיבלו ציון יותר גבוה בהסטוריה. ',
                H_COUNTER:1)
END.
```

תוכנית 9

```
PROGRAM MARKS (INPUT, OUTPUT);
VAR
   MARK, COUNTER, MAX_MARK, MIN_MARK, FAILS : INTEGER;
   AVERAGE : REAL;
BEGIN
   AVERAGE  := 0;
   FAILS    := 0;
   COUNTER  := 0;
   READLN (MARK);
   MAX_MARK := MARK;
   MIN_MARK := MARK;
   WHILE MARK <> -1 DO                         { -1 מסמן את סוף הקלט }
    BEGIN
      COUNTER := COUNTER + 1;                  { מספר הציונים }
      AVERAGE := AVERAGE + MARK;
      IF  MARK > MAX_MARK  THEN
         MAX_MARK := MARK;
      IF  MARK < MIN_MARK  THEN
         MIN_MARK := MARK;
      IF  MARK < 60  THEN
         FAILS := FAILS + 1;
      READLN (MARK)
    END;  { while }
   IF  COUNTER > 0 THEN
    BEGIN
      AVERAGE := AVERAGE / COUNTER;
      WRITELN (AVERAGE:4:2, 'ממוצע הציונים: ');
      WRITELN (FAILS:1,     'מספר הכשלונות: ');
      WRITELN (COUNTER:1,   'מספר הציונים: ');
      WRITELN (MAX_MARK:1,  'הציון הגבוה ביותר: ');
      WRITELN (MIN_MARK:1,  'הציון הנמוך ביותר: ')
    END
END.
```

__הסבר:__

MIN ו-MAX מקבלים את ערך הציון הראשון. כאשר הציונים הבאים שונים מהציון הראשון - ערכי MIN ו-MAX יתעדכנו בהתאם.

כאשר הציון הוא 1-, הקטע בשורות 13-23 לא מתבצע. אם 1- היה המספר הראשון אז COUNTER=0 - הקטע בשורות 25-32 לא יתבצע ולא יודפס דבר.

תוכנית 10

```
PROGRAM ANGLES (INPUT, OUTPUT);
VAR
   ANGLE : REAL;
BEGIN
   WRITELN ('הכנס את גודל הזוית במעלות');
   READLN (ANGLE);
   IF  (ANGLE >= 0) AND (ANGLE <= 360) THEN
    BEGIN
      IF  ANGLE = 360 THEN
         ANGLE := 0;
      WRITELN (TRUNC (ANGLE/90) + 1:1, ' מעלות נמצאת ברביע מספר'
         , ANGLE:4:2, 'זוית בגודל ')
    END
   ELSE
      WRITELN ('זוית שגוייה !')
END.
```

תוכנית 11

```
PROGRAM ANGLES (INPUT, OUTPUT);
VAR
   ANGLE : REAL;
BEGIN
   WRITELN ('הכנס את גודל הזוית במעלות');
   READLN (ANGLE);
   WHILE ANGLE >= 360  DO   ANGLE := ANGLE - 360;
   WRITELN (TRUNC (ANGLE / 90) + 1 :1, ' מעלות נמצאת ברביע מספר'
     , ANGLE:4:2, 'זוית בגודל ')
END.
```

הסבר:

כאשר הזוית גדולה מ- 360 מעלות, יש לחסר 360 מעלות מהזוית הנתונה, עד שמתקבלת זוית הקטנה מ- 360 מעלות.
זכור, כי זוית של 360 מעלות היא גם זוית בת 0 (אפס) מעלות. לדוגמה, זוית בת 789 מעלות שווה לזוית בת 69 מעלות.

תוכנית 12

התוכנית מתייחסת לתוכנית 7, פרק 3.

```
(1)  PROGRAM PRIME (INPUT, OUTPUT);
(2)  VAR
(3)     PRIME     : BOOLEAN;
(4)     INDEX, N : INTEGER;
(5)  BEGIN
(6)     WRITE ('N = ');
(7)     READLN (N);
(8)     INDEX := 2;
(9)     IF  N MOD INDEX >0  THEN
(10)       INDEX := INDEX + 1;
(11)    WHILE (INDEX < SQRT (N)) AND (N MOD INDEX > 0) DO
(12)       INDEX := INDEX + 2;
(13)    PRIME := NOT (N MOD INDEX = 0);        { TRUE אם המספר ראשוני }
(14)    IF PRIME THEN
(15)       WRITELN (' הוא ראשוני ', N:1, ' המספר')
(16)    ELSE
(17)       WRITELN (' איננו ראשוני ', N:1, ' המספר')
(18) END.
```

הסבר:

9 בדיקה האם המספר מתחלק ב- 2.
בדיקת החלוקה בשניים היא למעשה בדיקת החלוקה לכל המספרים הזוגיים.
אם N מתחלק ללא שארית למספר הזוגי הקטן ביותר (2), אז הוא מספר זוגי ואין צורך לבדוק את חלוקתו במספרים זוגיים אחרים.
אם המספר מתחלק ב-2 אז שורה 12 לא תתבצע, כי: N MOD INDEX = 0.
אם המספר איננו מתחלק ב-2, אז בדיקות החלוקה הבאות עד לשורש המספר נעשות על מספרים אי-זוגיים בלבד, כי ה-INDEX שהיה 2, הוגדל בערך אחד (שורה 10).

תוכנית 13

```
PROGRAM CARS (INPUT, OUTPUT);
VAR
   PASSENGERS, MAX_PASSENGERS,
   PASS_1, PASS_2, PASS_3, PASS_4, PASS_5, PASS_6,
   PER_1, PER_2, PER_3, PER_4, PER_5          : INTEGER;
BEGIN
   PASS_1 := 0;
   PASS_2 := 0;
   PASS_3 := 0;
   PASS_4 := 0;
   PASS_5 := 0;
   PASS_6 := 0;
   PER_1  := 0;
   PER_2  := 0;
   PER_3  := 0;
   PER_4  := 0;
   PER_5  := 0;
   READLN (PASSENGERS, MAX_PASSENGERS);
   WHILE  PASSENGERS <> 0  DO
    BEGIN
      CASE PASSENGERS OF
         1: PASS_1 := PASS_1 + 1;
         2: PASS_2 := PASS_2 + 1;
         3: PASS_3 := PASS_3 + 1;
         4: PASS_4 := PASS_4 + 1;
         5: PASS_5 := PASS_5 + 1;
         6: PASS_6 := PASS_6 + 1
      END;
      CASE TRUNC (PASSENGERS / MAX_PASSENGERS * 5) OF
         0    : PER_1 := PER_1 + 1;
         1    : PER_2 := PER_2 + 1;
         2    : PER_3 := PER_3 + 1;
         3    : PER_4 := PER_4 + 1;
         4, 5 : PER_5 := PER_5 + 1                    { 5 FOR 100% }
      END;
      READLN (PASSENGERS, MAX_PASSENGERS)
    END;    { while }
   WRITELN ('מספר נוסעים    מספר מכוניות ');
   WRITELN ('-------------------------------');
   WRITELN (PASS_1:10, 1:15);
   WRITELN (PASS_2:10, 2:15);
   WRITELN (PASS_3:10, 3:15);
   WRITELN (PASS_4:10, 4:15);
   WRITELN (PASS_5:10, 5:15);
   WRITELN (PASS_6:10, 6:15);
   WRITELN;
   WRITELN ('אחוז ניצולת    מספר מכוניות ');
   WRITELN ('-------------------------------');
   WRITELN (PER_1:10, ' 0-20%':15);
```

```
(50)     WRITELN (PER_2:10, ' 20-40%':15);
(51)     WRITELN (PER_3:10, ' 40-60%':15);
(52)     WRITELN (PER_4:10, ' 60-80%':15);
(53)     WRITELN (PER_5:10, ' 80-100%':15)
(54) END.
```

<u>הסבר:</u>

הנוסחה שבשורה 29 מיועדת להמיר את מספר הנוסעים במכונית במספר שלם המייצג את התחום של אחוז ניצולת המכונית. למשל, במקרה של 3 נוסעים נקבל את המספר 2 שמייצג את התחום 40%-60%.

תוכנית 15 א'

```
PROGRAM BEWARE_MAXINT (INPUT, OUTPUT);
CONST
   MAXINT = 32767.0;                        { REAL TYPE CONSTANT }
VAR
   A, B                          : INTEGER;
   A_REAL, B_REAL, AB_REAL : REAL;
BEGIN
   READLN (A, B);
   A_REAL := A;
   B_REAL := B;
   AB_REAL := A_REAL + B_REAL;
   IF  ABS (AB_REAL) > MAXINT  THEN
      IF  AB_REAL < 0  THEN
         WRITELN (ABS (AB_REAL) - MAXINT:1:0,
            'סכום המספרים קטן מ- -MAXINT ב:')
      ELSE
         WRITELN (ABS (AB_REAL) - MAXINT:1:0,
            'סכום המספרים גדול מ- MAXINT ב: ')
   ELSE
      WRITELN (A + B:1, 'סכום המספרים הוא ')
END.
```

תוכנית 15 ב'

```
(1)  PROGRAM BEWARE_MAXINT (INPUT, OUTPUT);
(2)  VAR
(3)     A, B :INTEGER;
(4)  BEGIN
(5)     READLN (A, B);
(6)     IF  B >= 0  THEN
(7)        IF  A > MAXINT - B  THEN
(8)           WRITELN (A - (MAXINT - B):1,
(9)              'סכום המספרים גדול מ- MAXINT ב: ')
(10)       ELSE
(11)          WRITELN (A + B:1, 'סכום המספרים הוא ')
(12)    ELSE   { B < 0 }
(13)       IF  A < - MAXINT - B  THEN
(14)          WRITELN ((-MAXINT - B) - A:1,
(15)             'סכום המספרים קטן מ- -MAXINT ב: ')
(16)       ELSE
(17)          WRITELN (A + B:1, 'סכום המספרים הוא ')
(18) END.
```

הסבר:

בשורות 8 ו-14 משתמשים בסוגריים כדי לקבוע את סדר הפעולות ובכך למנוע גלישה אפשרית.

תוכנית 16

תוכנית זו דומה לתוכנית 11. מוצע לפתור שאלה זו בעזרת מערכים לאחר הלימוד של נושא זה.

תוכנית 17

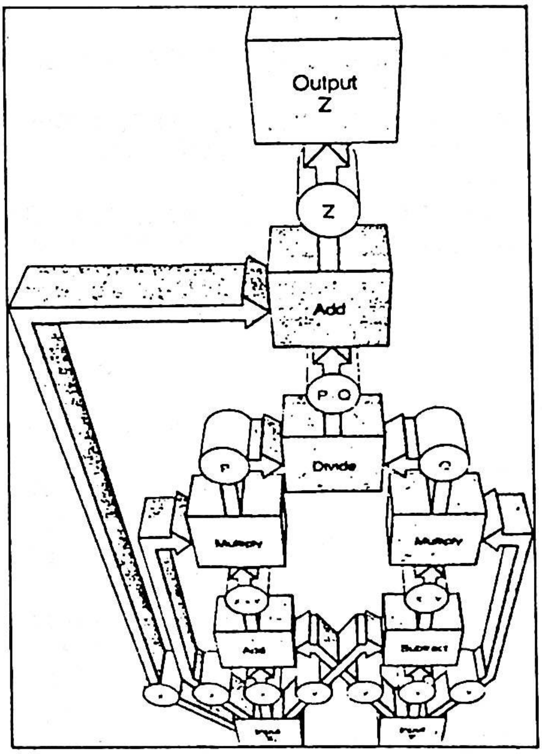
```
PROGRAM STUDENTS (INPUT, OUTPUT);
VAR
   A, B, C, YEAR, YEAR_3, YEAR_4 : REAL;
   INDEX                         : INTEGER;
BEGIN
   READLN (A, B, C);
   YEAR := A - C;
   B := 1 + B / 100;
   YEAR_4 := 0;
   FOR INDEX := 1 TO 4 DO
    BEGIN
      YEAR_4 := YEAR_4 + YEAR;
      YEAR := YEAR * B
    END;
```

```
    YEAR := A * 4 / 3;
    YEAR_3 := 0;
    FOR INDEX := 1 TO 3 DO
     BEGIN
       YEAR_3 := YEAR_3 + YEAR;
       YEAR := YEAR * B
     END;
    WRITELN (YEAR_4:4:2, 'העלות של מסלול א: ');
    WRITELN (YEAR_3:4:2, 'העלות של מסלול ב: ')
END.
```

תוכנית 18

```
PROGRAM DATES (INPUT, OUTPUT);
VAR
    A, B, C, D, E, F,
    DAY, MONTH, YEAR  : INTEGER;
BEGIN
    WRITELN ('הכנס יום חודש ושנה          ');
    READLN (DAY, MONTH, YEAR);
    A := YEAR MOD 100;
    B := A DIV 4;
    C := A + B;
    CASE MONTH OF
       1, 10     : D := 1;
       2, 3, 11  : D := 4;
       4, 7      : D := 0;
       5         : D := 2;
       6         : D := 5;
       8         : D := 3;
       9, 12     : D := 6
    END;
    E := C + D;
    F := E + DAY;
    F := F MOD 7;
    CASE F OF
       1 : WRITE ('ראשון');
       2 : WRITE  ('שני');
       3 : WRITE ('שלישי');
       4 : WRITE ('רביעי');
       5 : WRITE ('חמישי');
       6 : WRITE  ('שישי');
       0 : WRITE  ('שבת')
    END;
    WRITELN ('חל ביום ', DAY:1, '/', MONTH:1, '/', A:1, ' התאריך')
END.
```

```
PROGRAM NUMBERS (OUTPUT);
VAR
   INDEX, DIGIT_1, DIGIT_2, DIGIT_3 : INTEGER;
BEGIN
   FOR  INDEX := 1  TO  500   DO
    BEGIN
      DIGIT_1 := INDEX MOD 10;
      DIGIT_2 := (INDEX DIV 10) MOD 10;
      DIGIT_3 := (INDEX DIV 100) MOD 10;
      DIGIT_1 := DIGIT_1 * DIGIT_1 * DIGIT_1;
      DIGIT_2 := DIGIT_2 * DIGIT_2 * DIGIT_2;
      DIGIT_3 := DIGIT_3 * DIGIT_3 * DIGIT_3;
      IF  DIGIT_1 + DIGIT_2 + DIGIT_3 = INDEX THEN
         WRITELN (INDEX:3)
    END
END.
```



"פסקל - שפה לתכנות מבני"

פרק 5
מערכים

שאלות

שאלה 1

```
TYPE
   DAYS = (SUNDAY, MONDAY, TUESDAY, WEDNSDAY, THURSDAY, FRIDAY,
                                                    SATRDAY);
   COLORS = (YELLOW, ORANGE, RED, GREEN, BLUE, PURPLE);
   EDUCATION = (GAN, AMAMI, TICHON, HANDESI, UNIVERSITA );
   SEX = (MALE, FEMALE);
   MONTHS = (JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY,
                AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER);
```

שאלה 2

```
   DEGREE = 32..212;
   YEAR = 1900..1999;
   WORK_DAY = SUNDAY..THURSDAY;
```

שאלה 3

יש לשנות את פקודת ה-WRITE המוזכרת בשאלה לפקודת WRITELN, כדי שהפלט יראה כך:

התו ^ מציין רווח.

```
185^^^^^^^^^^^^1
113^^^^^^^^^^^^4
98^^^^^^^^^^^^5
85^^^^^^^^^^^^9
68^^^^^^^^^^^^12
```

תוכניות

תוכנית 1

```
PROGRAM VECTORS (INPUT, OUTPUT);
CONST
   MAX_INDEX = 100;
TYPE
   VECTOR = ARRAY [1..MAX_INDEX] OF REAL;
VAR
   A, B      : VECTOR;
   SUM       : REAL;
   INDEX, N  : INTEGER;
BEGIN
   SUM := 0;
   READLN (N);
   FOR  INDEX := 1  TO  N  DO
      READLN (A [INDEX]);
   FOR  INDEX := 1  TO  N  DO
    BEGIN
      READLN (B [INDEX]);
      SUM := SUM + A [INDEX] * B [INDEX]
    END;
   WRITELN (SUM:4:2, 'תוצאת המכפלה הפנימית היא ')
END.
```

תוכנית 2

```
PROGRAM VECTORS (INPUT, OUTPUT);
CONST
   MAX_INDEX = 100;
TYPE
   VECTOR = ARRAY [1..MAX_INDEX] OF REAL;
VAR
   A     : VECTOR;
   INDEX, N : INTEGER;
   SUM   : REAL;
BEGIN
   READLN (N);
   FOR  INDEX := 1  TO  N  DO
      READLN (A [INDEX]);
   SUM := A [1];
   FOR  INDEX := 2  TO  N  DO
    BEGIN
      SUM := SUM + A [INDEX];
      A [INDEX] := SUM - A [INDEX]
```

```
(16)      END
(17) END.
```

הסבר:

17 בכל שלב מקבלים ב-SUM יש את הסכום של האיבר הנוכחי ושל האיברים הקודמים לו בווקטור.

18 האיבר הנוכחי מקבל את הערך של SUM פחות ערכו העצמי, כלומר - את סכום האיברים הקודמים לו בלבד.

תוכנית 3

```
PROGRAM VECTORS (INPUT, OUTPUT);
CONST
   MAX_INDEX = 100;
TYPE
   VECTOR = ARRAY [1..MAX_INDEX] OF REAL;
VAR
   A                          : VECTOR;
   INDEX, LOCATION, N, M : INTEGER;
BEGIN
   WRITELN ('מספר אברי הוקטור?');
   READLN (N);
   WRITELN ('קרא את אברי הוקטור');
   FOR  INDEX := 1  TO  N  DO
      READ (A [INDEX]);
   WRITELN;
   M := 1;
   FOR  INDEX := 2  TO  N  DO
    BEGIN
      LOCATION := M;
      WHILE  (LOCATION > 0) AND (A [LOCATION] <> A [INDEX]) DO
         LOCATION := LOCATION - 1;
      IF  LOCATION = 0  THEN
       BEGIN
         M := M + 1;
         A [M] := A [INDEX]
       END
    END;
   FOR  LOCATION := 1 TO M DO
      WRITE (A [LOCATION]:4:2,'  ');
   WRITELN;
   WRITELN ('M = ', M:1)
END.
```

הסבר:

המשתנה M מצביע בכל שלב על האיבר בוקטור שעד אליו אין איברים כפולים.
המשתנה INDEX סורק את המערך בכל שלב. את האיבר ש-INDEX מצביע עליו משווים לאיברים מ-1 עד M (אין ביניהם כפולים כי בתהליך מתעלמים מהם) (שורות 19-21).

* אם האיבר מופיע בסריקה עד M, כלומר ב-M האיברים הראשונים, אז מתעלמים ממנו ו-INDEX מצביע לאיבר הבא אחרי זה שנפסל.
* אם המספר שעליו מצביע INDEX לא נמצא ב-M המספרים הראשונים, אז מגדילים את M (M:=M+1) והוא מועתק למקום ש-M מצביע עליו (שורות 24-25).

תוכנית 4

```
PROGRAM MAGIC (INPUT, OUTPUT);
CONST
   MAX_SIZE = 10;
TYPE
   MAGIC_SQUARE = ARRAY [1..MAX_SIZE, 1..MAX_SIZE] OF INTEGER;
VAR
   SQUARE                                   : MAGIC_SQUARE;
   INDEX_1, INDEX_2, N, SUM_1, SUM_2, SUM_3 : INTEGER;
BEGIN
   WRITELN ('מימד הריבוע?');
   READLN (N);
   FOR  INDEX_1 := 1  TO  N  DO
    BEGIN
      WRITELN (INDEX_1:1, ' הכנס שורה מספר');
      FOR  INDEX_2 := 1  TO  N  DO
         READ (SQUARE [INDEX_1, INDEX_2]);
      WRITELN
    END;
   SUM_1 := 0;
   SUM_2 := 0;
   FOR  INDEX_1 := 1  TO  N  DO
    BEGIN
      SUM_1 := SUM_1 + SQUARE [INDEX_1, INDEX_1];
      SUM_2 := SUM_2 + SQUARE [N + 1 - INDEX_1, INDEX_1]
    END;
   SUM_3 := SUM_1;
   INDEX_2 := 1;
   WHILE  (SUM_1 = SUM_2) AND
        (SUM_2 = SUM_3) AND (INDEX_2 <= N)  DO
    BEGIN
        SUM_1 := 0;
        SUM_2 := 0;
        FOR  INDEX_1 := 1  TO  N  DO
         BEGIN
```

```
(35)            SUM_1 := SUM_1 + SQUARE [INDEX_2, INDEX_1];
(36)            SUM_2 := SUM_2 + SQUARE [INDEX_1, INDEX_2]
(37)          END;     { for }
(38)         INDEX_2 := INDEX_2 + 1
(39)      END;       { while }
(40)     IF (SUM_1 = SUM_2) AND (SUM_2 = SUM_3)  THEN
(41)        WRITELN ('ריבוע זה הינו ריבוע קסם ')
(42)     ELSE
(43)        WRITELN ('ריבוע זה אינו ריבוע קסם ')
(44) END.
```

<u>הסבר:</u>

19-25 חישוב הסכום של שני האלכסונים.
* איברי האלכסון הראשון, (שורה 22): [1,1] [2,2] [3,3]...[n,n]
* איברי האלכסון השני (שורה 24):
[1,n] [2,n-1] [3,n-2]...[n-1,n-(n-1)+1] [n,1]

26 מעתה ואילך סכום איברי האלכסון נשמר ב- SUM_3.

27-39 משתנה INDEX_2 מורה על מספר השורה.
משתנה INDEX_1 (שורה 32) מורה על מספר העמודה.
סכום איברי השורה יחושב ב-SUM_1, סכום איברי העמודה יחושב ב-SUM_2.

כל עוד שלושת הסכומים (שורה, עמודה ואלכסון) שווים ומספר השורה קטן או שווה ל-N - מתבצעת הלולאה.

40-43 בדיקת יציאה מהלולאה. <u>אם</u> היציאה היתה לאחר שנסרקו כל השורות (יציאה בגלל INDEX_2), <u>אז</u> זהו ריבוע קסם.
<u>אם</u> היציאה מהלולאה היתה בגלל אי-שיוויון בסכומים, <u>אז</u> הבדיקה בשורה 39 תביא לביצוע שורה 42, והמסקנה היא שהריבוע אינו ריבוע קסם.

תוכנית 5

```
PROGRAM LATIN (INPUT, OUTPUT);
CONST
   MAX_SIZE = 10;
TYPE
   LATIN_SQUARE = ARRAY [1..MAX_SIZE, 1..MAX_SIZE] OF INTEGER;
VAR
   SQUARE        : LATIN_SQUARE;
   INDEX_1, INDEX_2, INDEX_3, INDEX_4, N : INTEGER;
   OK : BOOLEAN;
BEGIN
   WRITELN ('מימד הריבוע?');
   READLN (N);
```

```
    FOR  INDEX_1 := 1  TO  N  DO
     BEGIN
       WRITELN (INDEX_1:1, 'הכנס שורה מספר ');
       FOR  INDEX_2 := 1  TO  N  DO
          READ (SQUARE [INDEX_1, INDEX_2]);
       WRITELN
     END;
    OK := TRUE;
    FOR  INDEX_1 := 1  TO  N  DO
       FOR  INDEX_2 := 1  TO  N  DO
          FOR  INDEX_3 := INDEX_1 + 1  TO  N  DO
             FOR  INDEX_4 := INDEX_2 + 1  TO  N  DO
                IF (SQUARE [INDEX_1, INDEX_2] =
                                          SQUARE [INDEX_3, INDEX_2])
                   OR (SQUARE [INDEX_1, INDEX_2] =
                                          SQUARE [INDEX_1, INDEX_4])
                THEN
                   OK := FALSE;
    IF  OK  THEN
       WRITELN ('ריבוע זה הוא ריבוע לטיני ')
    ELSE
       WRITELN ('ריבוע זה אינו ריבוע לטיני ')
END.
```

תוכנית 6

```
PROGRAM CALENDAR (INPUT, OUTPUT);
CONST
   FIRST_1986_DAY = 4;          { היום הראשון בשנת 1986 היה יום רביעי }
VAR
   YEAR  : 1986..2000;
   MONTH : 1..12;
   DAYS  : 28..31;
   I     : INTEGER;
   FIRST_DAY : 1..7;
BEGIN
   WRITE ('YEAR? ');
   READLN (YEAR);
   FIRST_DAY := FIRST_1986_DAY;
   FOR  I := 1987  TO  YEAR  DO
     BEGIN
       FIRST_DAY := FIRST_DAY MOD 7 + 1;  { שנה רגילה: 365 MOD 7 = 1 }
       IF  I MOD 4 = 0  THEN
          FIRST_DAY := FIRST_DAY MOD 7 + 1 { בשנה מעוברת ישנם 366 יום}
     END;
   FOR  MONTH := 1  TO  12  DO
     BEGIN
      WRITELN;
```

```
  CASE  MONTH  OF                                      { הדפסת שם החודש }
     1  : WRITE ('JANUARY':35);
     2  : WRITE ('FEBRUARY':35);
     3  : WRITE ('MARCH':35);
     4  : WRITE ('APRIL':35);
     5  : WRITE ('MAY':35);
     6  : WRITE ('JUNE':35);
     7  : WRITE ('JULY':35);
     8  : WRITE ('AUGUST':35);
     9  : WRITE ('SEPTEMBER':35);
     10 : WRITE ('OCTOBER':35);
     11 : WRITE ('NOVEMBER':35);
     12 : WRITE ('DECEMBER':35)
  END;
  WRITELN (YEAR:8);                                         { הדפסת השנה }
  WRITELN ('--------------------':45);
  WRITELN
     ('     SUN     MON     TUE     WED     THU     FRI     SAT');
  WRITELN
    ('    -----   -----   -----   -----   -----   -----   -----');
  CASE  MONTH  OF
     1, 3, 5, 7, 8, 10, 12 : DAYS := 31;     { חודשים שבהם 31 יום }
     4, 6, 9, 11           : DAYS := 30;     { חודשים שבהם 30 יום }
     2                     : IF  YEAR MOD 4 = 0  THEN
                                DAYS := 29   { בשנה מעוברת בחודש }
                                             { פברואר יש 29 יום. }
                             ELSE
                                DAYS := 28 { בשנה רגילה יש בחודש }
                                             { פברואר יש 28 יום }
  END;
  FOR  I := 1  TO  FIRST_DAY + DAYS - 1  DO
    BEGIN
     IF  I < FIRST_DAY  THEN
        WRITE (' ':8)    { הדפסת רווחים עד היום שבו חל ה-1 לחודש }
     ELSE
        WRITE (I - FIRST_DAY + 1 :8); { הדפסת התאריך ביום המתאים }
     IF  I MOD 7 = 0  THEN
        WRITELN                { נגמר שבוע (שורה), מעבר לשבוע חדש }
    END;
  WRITELN;
  FIRST_DAY := (FIRST_DAY + DAYS - 1) MOD 7 + 1        { חישוב היום }
                                                  { הראשון בחודש הבא }
 END  { for month }
END.
```

<u>הסבר:</u>

בשנה רגילה יש 365 יום, 365 MOD 7 = 1, כלומר היום הראשון של השנה הנוכחית, הוא היום בו חל היום הראשון של השנה הקודמת + 1.
בשנה מעוברת ישנם 366 יום, לכן 2 = 366 MOD 7, כלומר היום הראשון של השנה הנוכחית, הוא היום בו חל היום הראשון של השנה הקודמת + 2.

תוכנית 7 א'

```
PROGRAM MATRIX_A (INPUT, OUTPUT);
CONST
   MAX_SIZE = 20;
TYPE
   MATRIX = ARRAY [1..MAX_SIZE, 1..MAX_SIZE] OF REAL;
VAR
   A                    : MATRIX;
   INDEX_1, INDEX_2, N  : INTEGER;
   OK                   : BOOLEAN;
BEGIN
   WRITELN ('מהו גודל המטריצה?');
   READLN (N);
   FOR  INDEX_1 := 1  TO  N  DO
    BEGIN
      WRITELN (INDEX_1:1, 'הכנס שורה מספר ');
      FOR  INDEX_2 := 1  TO  N  DO
         READ (A [INDEX_1, INDEX_2]);
      WRITELN
    END;
   OK := TRUE;
   FOR  INDEX_1 := 1  TO  N  DO
      FOR  INDEX_2 := INDEX_1  TO  N  DO
         IF  A [INDEX_1, INDEX_2] <> A [INDEX_2, INDEX_1] THEN
            OK := FALSE;
   IF  OK  THEN
      WRITELN ('מטריצה זו היא מטריצה סימטרית ')
   ELSE
      WRITELN ('מטריצה זו איננה סימטרית ')
END.
```

תוכנית 7 ב'

```
PROGRAM MATRIX_B (INPUT, OUTPUT);
CONST
   MAX_SIZE = 20;
TYPE
   MATRIX = ARRAY [1..MAX_SIZE, 1..MAX_SIZE] OF REAL;
   INDEX  = 1..MAX_SIZE;
VAR
   A, B, C                                : MATRIX;
   INDEX_1, INDEX_2, INDEX_3, N, M, K : INDEX;
BEGIN
   WRITE ('number of coloums in A, rows in B   N = ');
   READLN (N);
   WRITE ('number of rows in A   M = ');
   READLN (M);
```

```
    WRITE ('number of coloums in B   K = ');
    READLN (K);
    WRITELN ('הכנסת נתונים של מטריצה A ');
    FOR  INDEX_1 := 1  TO  M  DO
     BEGIN
       WRITELN (INDEX_1:1, 'הכנס שורה מספר ');
       FOR  INDEX_2 := 1  TO  N  DO
          READ (A [INDEX_1, INDEX_2]);
       WRITELN
     END;
    WRITELN ('הכנסת נתונים של מטריצה B ');
    FOR  INDEX_1 := 1  TO  N  DO
     BEGIN
       WRITELN (INDEX_1:1, 'הכנס שורה מספר ');
       FOR  INDEX_2 := 1  TO  K  DO
          READ (B [INDEX_1, INDEX_2]);
       WRITELN
     END;
    WRITELN ('תוצאת המכפלה של A ו-B היא: ');
    FOR  INDEX_1 := 1  TO  M  DO                     { לולאה על שורות A }
     BEGIN
       FOR  INDEX_2 := 1  TO  K  DO                  { לולאה על עמודות B }
        BEGIN
          C [INDEX_1, INDEX_2] := 0;                 { ערך התחלתי לתוצאה }
                                                     { מכפלה פנימית של     }
          FOR  INDEX_3 := 1  TO  N  DO               { שורה ב-A בעמודה ב-B }
             C [INDEX_1, INDEX_2] := C [INDEX_1, INDEX_2] +
                      A [INDEX_1, INDEX_3] * B [INDEX_3, INDEX_2];
          WRITE (C [INDEX_1, INDEX_2]:8:2, '  ')     { הדפסת התוצאה }
        END;
       WRITELN
     END { for index_1 }
END.
```

<u>הסבר:</u>

* מכפלת המטריצות A ב-B (A*B) מוגדרת רק כאשר מתקיים התנאי הבא:

מספר העמודות במטריצה A שווה למספר השורות במטריצה B.

* גודל המטריצה C המקיימת (A*B=C) הוא:
  * מספר השורות הוא כמספר השורות במטריצה A.
  * מספר העמודות הוא כמספר העמודות במטריצה B.

$$A_{(m,n)} * B_{(n,k)} = C_{(m,k)}$$

תוכנית 9

```
PROGRAM RAIN (INPUT, OUTPUT);
TYPE
   MONTHS = (JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY,
             AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER);
VAR
   INDEX, MONTH, MAX_MONTH, MIN_MONTH, MAX_SECOND_MONTH : MONTHS;
   RAIN                          : ARRAY [MONTHS] OF REAL;
   MAX, SECOND_MAX, MIN, AVERAGE : REAL;
BEGIN
   MAX       := -9999;                          { ערכים התחלתיים }
   MAX_MONTH := JANUARY;
   MIN       := 9999;
   MIN_MONTH := JANUARY;
   AVERAGE   := 0;
   FOR  MONTH := JANUARY  TO  DECEMBER  DO
    BEGIN
      READ (RAIN [MONTH]);
      IF  RAIN [MONTH] > MAX THEN
       BEGIN
         MAX_MONTH := MONTH;
         MAX       := RAIN [MONTH]
       END
      ELSE
         IF  RAIN [MONTH] < MIN THEN
          BEGIN
            MIN_MONTH := MONTH;
            MIN       := RAIN [MONTH]
          END;
      AVERAGE := AVERAGE + RAIN [MONTH] / 12
    END;
   WRITELN;                         { מציאת החודש השני הגשום ביותר }
   SECOND_MAX := MIN;
   MAX_SECOND_MONTH := MIN_MONTH;
   FOR  MONTH := JANUARY  TO  DECEMBER  DO
    BEGIN
      IF  (RAIN [MONTH] > SECOND_MAX) AND (MONTH <> MAX_MONTH)
      THEN  BEGIN
              MAX_SECOND_MONTH := MONTH;
              SECOND_MAX       := RAIN [MONTH]
             END
    END;
   WRITELN
      ('חודש      כמות הגשם      סטיה מהממוצע      הערות      ');
   FOR  MONTH := JANUARY  TO  DECEMBER  DO
    BEGIN
      IF  MONTH = MAX_MONTH  THEN
         WRITE ('הגשום ביותר':20)
      ELSE
         IF  MONTH = MIN_MONTH  THEN
```

```
            WRITE ('כמות גשם מינימלית':20)
         ELSE
            IF  MONTH = MAX_SECOND_MONTH  THEN
               WRITE ('השני הגשום ביותר':20)
            ELSE
               WRITE (' ':20);
      WRITE (RAIN [MONTH] - AVERAGE :10:2);
      WRITE (RAIN [MONTH]:12:2, '    ');
      CASE  MONTH  OF
         JANUARY   : WRITELN ('  ינואר');
         FEBRUARY  : WRITELN (' פברואר');
         MARCH     : WRITELN ('    מרץ');
         APRIL     : WRITELN ('  אפריל');
         MAY       : WRITELN ('    מאי');
         JUNE      : WRITELN ('   יוני');
         JULY      : WRITELN ('   יולי');
         AUGUST    : WRITELN (' אוגוסט');
         SEPTEMBER: WRITELN (' ספטמבר');
         OCTOBER   : WRITELN ('אוקטובר');
         NOVEMBER  : WRITELN (' נובמבר');
         DECEMBER  : WRITELN ('  דצמבר')
      END
     END;  { for month }
    WRITELN;
    WRITELN (AVERAGE:8:2, 'כמות הגשם הממוצעת בשנה היא: ')
END.
```

תוכנית 10

נוסחת מקדם המתאם:

$$r = \frac{\sum XY - n\bar{X}\bar{Y}}{\sqrt{[\sum X^2 - n\bar{X}^2][\sum Y^2 - n\bar{Y}^2]}}$$

```
PROGRAM ILL_DAYS (INPUT, OUTPUT);
CONST
   MAX_TESTS = 500;
TYPE
   AGE   = 1..120;
   DAYS  = 0..365;
   TESTS = 1..MAX_TESTS;
VAR
   N, TEST  : TESTS;
   X : ARRAY [TESTS] OF AGE;
   Y : ARRAY [TESTS] OF DAYS;
   EX, EX2, EY, EY2, EXY, R  : REAL;
BEGIN
   WRITE ('מספר התצפיות N = ');
   READLN (N);
```

```
   EX  := 0;
   EX2 := 0;
   EY  := 0;
   EY2 := 0;
   EXY := 0;
   WRITELN ('ימי מחלה   גיל');
   FOR  TEST := 1  TO  N  DO
      READLN (X [TEST], Y [TEST]);
   FOR  TEST := 1  TO  N  DO
    BEGIN
      EX := EX + X [TEST];
      EY := EY + Y [TEST];
      EX2 := EX2 + SQR (X [TEST]);
      EY2 := EY2 + SQR (Y [TEST]);
      EXY := EXY + X [TEST] * Y [TEST]
    END;
   R := (N * EXY - EX * EY) /
      (N * SQRT ((EX2 - SQR (EX) ) * (EY2 - SQR (EY) )));
   WRITELN (R:12:2, 'מקדם המתאם הוא: ')
END.
```

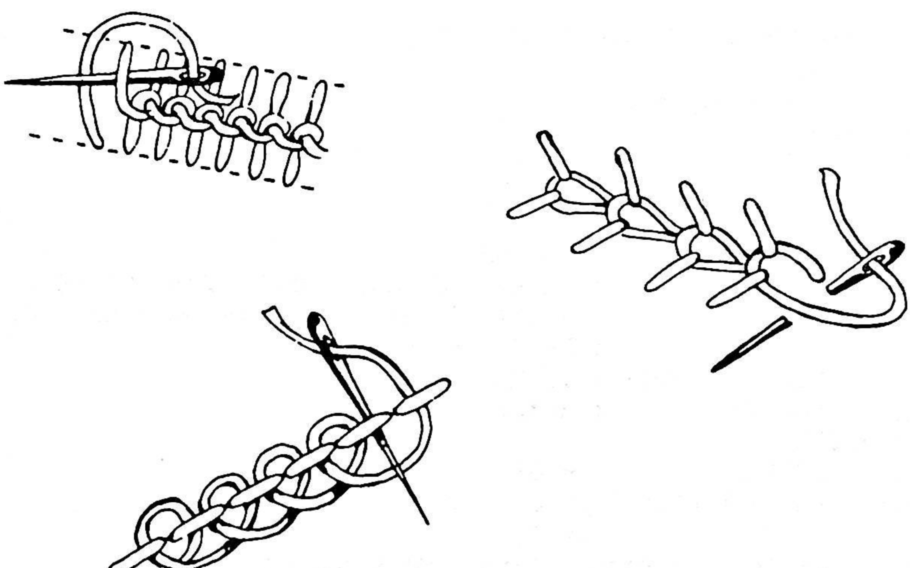
תוכנית 11

```
PROGRAM SALES (INPUT, OUTPUT);
CONST
   MAX_PARTS = 100;
TYPE
   MONTHS      = 1..12;
   PART_NUMBER = 0..9999;
VAR
   MONTH          : MONTHS;
   SALE           : ARRAY [MONTHS,1..MAX_PARTS] OF REAL;
   PARTS          : ARRAY [0..MAX_PARTS] OF PART_NUMBER;
   PART           : PART_NUMBER;
   SERIAL, INDEX  : 1..MAX_PARTS;
   AMOUNT         : REAL;
BEGIN
   SERIAL         := 0;
   PARTS [SERIAL] := 0;
   PART           := 1;
   WRITELN ('מספר פריט   כמות   חודש ');
   READLN (MONTH, AMOUNT, PART);
   WHILE  PART > 0  DO
    BEGIN
      IF  PART <> PARTS [SERIAL]  THEN
       BEGIN
         SERIAL := SERIAL + 1;
         FOR  INDEX := 1  TO  12  DO
```

```
(26)              SALE [INDEX, SERIAL] := 0
(27)         END;
(28)        PARTS [SERIAL] := PART;
(29)        SALE [MONTH, SERIAL] := SALE [MONTH, SERIAL] + AMOUNT;
(30)        READLN (MONTH, AMOUNT, PART)
(31)      END;
(32)     WRITELN;
(33)     WRITELN ('   מאי   יוני   יולי   אוג   ספט   אוק   נוב   דצמ   
(34)        'אפר   מרץ   פבר   ינו   פריט');
(35)     FOR  INDEX := 1  TO  SERIAL  DO
(36)      BEGIN
(37)        FOR  MONTH := 12  DOWNTO  1  DO
(38)           WRITE (SALE [MONTH, INDEX]:6:0);
(39)        WRITELN (PARTS [INDEX]:6)
(40)      END
(41) END.
```

<u>הערה:</u>

שורה 33 פוצלה כאן, כמו בשאלות קודמות בגלל מגבלות של אורך שורה בתוכנית הכתובה.



"לולאות"

פרק 6
נתונים לא מספריים

שאלות

שאלה 1

```
A ROSE
  IS
A ROSE
  IS
A ROSE
  IS
A ROSE
  IS
A ROSE
```

שאלה 2

```
MITZ         618
```

שאלה 3

```
PROGRAM CHARACTERS (INPUT, OUTPUT);
VAR
   I, OCT : INTEGER;
BEGIN
   FOR  I := 32  TO  127  DO
     BEGIN
      OCT := (I DIV 64) * 100 + ((I MOD 64) DIV 8) * 10 + I MOD 8;
      WRITELN (I, ' ':4, CHR (I), OCT:10)
     END
END.
```

הערה:

תוכנית זו להפיכה של מספרים מייצוג עשרוני לאוקטלי, מתאימה רק למספרים עשרוניים הקטנים מ- 512.

צורה כללית יותר של תוכנית זו תמצא בתוכנית 12 בפרק 8.

שאלה 4

```
PROGRAM KRIA (INPUT, OUTPUT);
VAR
   TAV    : CHAR;
   MISPAR : INTEGER;
BEGIN
   MISPAR := 0;
   READ (TAV);
   WHILE  TAV <> ' '  DO              { הקריאה תתבצע עד קריאת תו רווח }
     BEGIN
      IF  TAV <> ',' THEN
         MISPAR := 10 * MISPAR + ORD (TAV) - ORD ('0');
      READ (TAV)
     END;
   WRITELN (MISPAR)
END.
```

הסבר:

לסיפרה הראשונה מתייחסים כסיפרת אחדות. אם לאחריה מופיעה סיפרה נוספת, אז כופלים את סיפרת האחדות הנוכחית ב-10 ומוסיפים את הסיפרה החדשה כסיפרת אחדות. אם קוראים סיפרה נוספת, היא תגרום להכפלת הערך הנוכחי ב-10 והיא תהיה עתה סיפרת האחדות במספר וכך הלאה - עד אשר קוראים תו רווח ומסיימים.

שאלה 5

```
PROGRAM KTIVA (INPUT, OUTPUT);
VAR
   MISPAR,
   I, J   : INTEGER;
   MARACH : ARRAY [1..10] OF CHAR;
BEGIN
   WRITELN ('הקש מספר');
   READLN (MISPAR);
   I := 1;
   WHILE (MISPAR <> 0) AND (I <= 10) DO
    BEGIN
      MARACH [I] := CHR ( ORD ('0') + MISPAR MOD 10);
      MISPAR := MISPAR DIV 10;
```

```
      IF (MISPAR > 0) AND (I MOD 3 = 0) THEN
        BEGIN
          MARACH [I+1] := ',';
          I := I + 2
        END
      ELSE
          I := I + 1
     END;    { while }
   J := I - 1;
   FOR  I := J DOWNTO 1 DO
      WRITE (MARACH [I])
END.
```

תוכניות

תוכנית 1

```
PROGRAM TEXT (INPUT, OUTPUT);
VAR
   CHARACTER : CHAR;
BEGIN
   READ (CHARACTER);
   WHILE (CHARACTER >= 'A') AND (CHARACTER <= 'Z') DO
    BEGIN
      IF  CHARACTER = 'Z'  THEN
         WRITE ('A')
      ELSE
         WRITE (SUCC (CHARACTER));
      READ (CHARACTER)
    END
 END.
```

הסבר:

התוכנית פועלת רק על אותיות לטיניות גדולות.
יש לטפל במקרה המיוחד של מעבר מ-Z ל-A, שאר המעברים נעשים בעזרת הפונקציה SUCC.

תוכנית 2

```
(1)  PROGRAM TEXT (INPUT, OUTPUT);
(2)  CONST
(3)     MAX_NAME_LENGTH = 25;
(4)  TYPE
(5)     NAME_STRING = PACKED ARRAY [1..MAX_NAME_LENGTH] OF CHAR;
(6)  VAR
(7)     NAME  : NAME_STRING;
(8)     INDEX : INTEGER;
(9)  BEGIN
(10)    READLN (NAME);
(11)    WHILE  NAME [1] <> '.'  DO
(12)     BEGIN
(13)       INDEX := MAX_NAME_LENGTH;
(14)       WHILE  NAME [INDEX] = ' ' DO
(15)          INDEX := INDEX - 1;
(16)       WHILE  INDEX > 0 DO
(17)        BEGIN
(18)          WRITE (NAME [INDEX]);
(19)          INDEX := INDEX - 1
(20)        END;
(21)       WRITELN;
(22)       READLN (NAME)
(23)     END
(24) END.
```

<u>הסבר:</u>

11 סיום הקלט על ידי מלה שיש בה נקודה במקום הראשון (תמליל באנגלית).
13-15 חיפוש אחר סיום השם מסוף המערך.
16-20 הדפסת השם בסדר הפוך.

תוכנית 3

```
PROGRAM TEXT (INPUT, OUTPUT);
CONST
   MAX_NAME_LENGTH = 40;
TYPE
   NAME_STRING = PACKED ARRAY [1..MAX_NAME_LENGTH] OF CHAR;
VAR
   NAME                : NAME_STRING;
   NAME_LENGTH, INDEX, WEIGHT, HEIGHT : INTEGER;
   AVERAGE             : REAL;
BEGIN
   READLN (NAME);
   WHILE  NAME [1] <> '.'  DO
```

```
(13)       BEGIN
(14)         INDEX := 1;
(15)         WHILE ((NAME [INDEX] > '9') OR (NAME [INDEX] < '0'))
(16)                           AND (INDEX < MAX_NAME_LENGTH) DO
(17)          BEGIN
(18)            IF  NAME [INDEX] <> ' '  THEN
(19)               NAME_LENGTH := INDEX;
(20)            INDEX := INDEX + 1
(21)          END;
(22)         HEIGHT := 0;
(23)         WHILE (NAME [INDEX] <= '9') AND (NAME [INDEX] >= '0')
(24)                           AND (INDEX < MAX_NAME_LENGTH) DO
(25)          BEGIN
(26)            HEIGHT := 10 * HEIGHT + ORD (NAME [INDEX]) - ORD ('0');
(27)            INDEX := INDEX + 1
(28)          END;
(29)         WHILE (NAME [INDEX] = ' ') AND (INDEX < MAX_NAME_LENGTH) DO
(30)            INDEX := INDEX + 1;
(31)         WEIGHT := 0;
(32)         WHILE (NAME [INDEX] <= '9') AND (NAME [INDEX] >= '0')
(33)                           AND (INDEX < MAX_NAME_LENGTH) DO
(34)          BEGIN
(35)            WEIGHT := 10 * WEIGHT + ORD (NAME [INDEX]) - ORD ('0');
(36)            INDEX := INDEX + 1
(37)          END;
(38)         IF  INDEX < MAX_NAME_LENGTH  THEN
(39)          BEGIN
(40)            AVERAGE := WEIGHT / HEIGHT;
(41)            FOR INDEX:=1 TO NAME_LENGTH DO WRITE (NAME [INDEX]);
(42)            WRITELN (WEIGHT:15, HEIGHT:15, AVERAGE:15:2)
(43)          END;
(44)         READLN (NAME)
(45)       END
(46) END.
```

<u>הסבר:</u>

שורת הקלט נקראת למשתנה NAME כמחרוזת של תווים, ויש לפצל אותה למרכיביה השונים:

14-21 הפרדת השם.

22-28 הפרדת התווים הקשורים לגובה וחישוב הערך המספרי.

29-30 דילוג על רווחים אם ישנם.

31-37 הפרדה וחישוב המשקל.

38 פקודת התנאי מיועדת לבדוק אם שלושת נתוני הקלט מתקיימים. אם התנאים מתקיימים - מתחילים בחישוב (שורות 39-43), אחרת - התוכנית מתעלמת משורת הקלט הנוכחית, כי זוהי שורה שגויה.

תוכנית 5

```
(1)  PROGRAM TEXT (INPUT, OUTPUT);
(2)  CONST
(3)     MAX_WORD_LENGTH = 25;
(4)  TYPE
(5)     WORD_STRING = PACKED ARRAY [0..MAX_WORD_LENGTH] OF CHAR;
(6)  VAR
(7)     WORD      : WORD_STRING;
(8)     INDEX, I : INTEGER;
(9)  BEGIN
(10)    REPEAT
(11)       FOR  I := 0  TO  MAX_WORD_LENGTH  DO
(12)          WORD [I] := ' ';
(13)       READ (WORD [1]);
(14)       INDEX := 1;
(15)       WHILE  (WORD [INDEX] <> ' ') AND (WORD [INDEX] <> '.')  DO
(16)        BEGIN
(17)          INDEX := INDEX + 1;
(18)          READ (WORD [INDEX])
(19)        END;
(20)       I := 1;
(21)       WHILE (I < INDEX) AND NOT ((WORD [I] = 'E') AND
(22)          (WORD [I+1] = 'I') AND (WORD [I-1] <>'C')) AND NOT
(23)          ((WORD [I] = 'I') AND (WORD [I+1] = 'E') AND
(24)          (WORD [I-1] = 'C')) DO
(25)          I := I + 1;
(26)       IF  I < INDEX  THEN
(27)        BEGIN
(28)          WRITELN;
(29)          WRITELN ('לא מתאימה לכלל', WORD)
(30)        END
(31)    UNTIL  WORD [INDEX] = '.'
(32) END.
```

<u>הסבר:</u>

13-19 הקריאה מתבצעת מלה אחר מלה, תוך הנחה שאורך המלה הנקראת קטן מהערך שהוגדר ב-CONST.
כדי לשכלל זאת - ניתן להוסיף את התנאי הבא בשורה 15:
AND (INDEX < MAX_WORDLENGHT)

21-24 התנאי בודק אם אין סידור אותיות שסותר את הכלל.

26 <u>אם</u> I<INDEX (כלומר לא עברנו את כל המלה) <u>אז</u> ישנה סתירה לכלל ולכך יש הודעה מתאימה (שורה 29). <u>אחרת</u> - ממשיכים למלה הבאה.

31 תנאי סיום התוכנית הוא נקודה.

תוכנית 6

```
PROGRAM POST (INPUT, OUTPUT);
TYPE
   DESTINATION = (M, E, K);
   WEIGHT      = (KL, MM, C);
   SERVICE     = (EX, R);
   SEND_COST   = (POINT, LIMIT, PER_KG);
   TABLE =
      ARRAY [DESTINATION, WEIGHT, SERVICE, SEND_COST] OF INTEGER;
VAR            {  יעד        משקל     שירות    עלות משלוח }
   PRICE : TABLE;
   DEST  : DESTINATION;
   WEIG  : WEIGHT;
   SERV  : SERVICE;
   COST  : SEND_COST;
   CH    : CHAR;
   KG, DIFF_KG, VALUE, TOTAL : REAL;
BEGIN
   FOR  DEST := M  TO  K  DO
      FOR  WEIG := KL  TO  C  DO
         FOR  SERV := EX  TO  R  DO
            FOR  COST := POINT  TO  PER_KG  DO
               PRICE [DEST, WEIG, SERV, COST] := 0;
   PRICE [M, KL, EX, POINT]  := 12;
   PRICE [M, KL,  R, POINT]  := 6;
   PRICE [M, MM, EX, PER_KG] := 3;
   PRICE [M, MM,  R, PER_KG] := 2;
   PRICE [M,  C, EX, POINT]  := 100;
   PRICE [M,  C, EX, PER_KG] := 2;
   PRICE [M,  C, EX, LIMIT]  := 20;
   PRICE [M,  C,  R, POINT]  := 50;
   PRICE [M,  C,  R, PER_KG] := 2;
   PRICE [M,  C,  R, LIMIT]  := 20;
   PRICE [E, KL, EX, POINT]  := 12;
   PRICE [E, KL,  R, POINT]  := 6;
   PRICE [E, MM, EX, PER_KG] := 6;
   PRICE [E, MM,  R, PER_KG] := 3;
   PRICE [E,  C, EX, PER_KG] := 6;
   PRICE [E,  C,  R, POINT]  := 3;
   PRICE [K, KL, EX, POINT]  := 12;
   PRICE [K, KL,  R, POINT]  := 12;
   PRICE [K, MM, EX, PER_KG] := 8;
   PRICE [K, MM,  R, PER_KG] := 6;
   PRICE [K,  C, EX, POINT]  := 100;
   PRICE [K,  C, EX, PER_KG] := 2;
   PRICE [K,  C, EX, LIMIT]  := 20;
   PRICE [K,  C,  R, POINT]  := 50;
   PRICE [K,  C,  R, PER_KG] := 2;
   PRICE [K,  C,  R, LIMIT]  := 20;
   WRITELN ('מקום היעד: M - מקומי, E - עיר גדולה, K - עיר קטנה');
```

```
(50)     READLN (CH);
(51)     CASE CH OF
(52)        'M' : DEST := M;
(53)        'E' : DEST := E;
(54)        'K' : DEST := K
(55)      END;
(56)     WRITELN ('משקל בקג');
(57)     READLN (KG);
(58)     IF  KG < 1  THEN
(59)        WEIG := KL
(60)     ELSE
(61)        IF  KG <= 10  THEN
(62)           WEIG := MM
(63)        ELSE
(64)           WEIG := C;
(65)     WRITELN ('שרות:  R - רגיל, E - אקספרס');
(66)     READLN (CH);
(67)     CASE CH OF
(68)        'R' : SERV := R;
(69)        'E' : SERV := EX
(70)      END;
(71)     WRITELN ('הערך השקלי של נקודה');
(72)     READLN (VALUE);
(73)     IF  PRICE [DEST, WEIG, SERV, LIMIT] < KG  THEN
(74)        DIFF_KG := KG - PRICE [DEST, WEIG, SERV, LIMIT]
(75)     ELSE
(76)        DIFF_KG := 0;
(77)     TOTAL := VALUE * (PRICE [DEST, WEIG, SERV, POINT] +
(78)                 PRICE [DEST, WEIG, SERV, PER_KG] * DIFF_KG);
(79)     WRITELN (TOTAL:4:2, ' המחיר למשלוח החבילה הוא: ')
(80) END.
```

<u>הסבר:</u>

בתוכנית זו משתמשים במערך ארבעת ממדים בכדי להחסן את הטבלה.
האינדקסים הם:

1. מקום היעד.
2. משקל.
3. סוג השירות.
4. SEND_COST מכיל 3 אלמנטים: את מספר הנקודות (POINT), את גבול המשקל שמעליו יש להוסיף עוד נקודות (LIMIT), ואת מספר הנקודות לק"ג מעבר לגבול זה (PER_KG).

18-48 איפוס המערך והכנסת הטבלה.

49-72 קריאת הנתונים ועיבודם.

73-78 חישוב המחיר לפי הטבלה. אופן החישוב: מציאת המשקל העודף (אם יש), והכפלתו במספר הנקודות לק"ג. לאחר מכן הוספה למספר הנקודות הקבוע והכפלת התוצאה בערך של כל נקודה.

תוכנית 7

```
PROGRAM TEXT (INPUT, OUTPUT);
CONST
   MAX_LINE_LENGTH = 70;
   MAX_LINES       = 20;
TYPE
   TEXT_LINE = PACKED ARRAY [1..MAX_LINE_LENGTH] OF CHAR;
   TEXT_PAGE = ARRAY [1..MAX_LINES] OF TEXT_LINE;
VAR
   LINE     : TEXT_LINE;
   PAGE     : TEXT_PAGE;
   INDEX, SPACE_NU, COUNTER, I, PTR : INTEGER;
   NO_ERROR            : BOOLEAN;
BEGIN
   INDEX := 0;
   REPEAT
      INDEX := INDEX + 1;
      READLN (PAGE [INDEX])
   UNTIL  PAGE [INDEX, 1] = '.';
   FOR  I := 1  TO  INDEX  DO
      IF  PAGE [I][1] = '#'  THEN
       BEGIN
         NO_ERROR := TRUE;
         SPACE_NU := 0;
         PTR := 2;          { מצביע על המיקום בשורה לאחר ה- '#' }
         WHILE (PAGE [I][PTR] <= '9') AND
               (PAGE [I][PTR] >= '0') DO
          BEGIN
           SPACE_NU := 10 * SPACE_NU + ORD (PAGE [I][PTR]) -
                                                      ORD ('0');
           PTR := PTR + 1
          END;
         IF (PTR = 2) OR (SPACE_NU < 1) OR (SPACE_NU > 120) OR
            (PAGE [I][PTR] <> ' ') THEN
              WRITELN (PAGE [I])
         ELSE
            BEGIN
              FOR  COUNTER := 1  TO  SPACE_NU  DO
                 WRITE (' ');
              COUNTER := 70;
              WHILE  PAGE [I][COUNTER] = ' '  DO
                 COUNTER := COUNTER - 1;
              WHILE  PTR < COUNTER  DO
               BEGIN
                 PTR := PTR + 1;
                 WRITE (PAGE [I][PTR])
               END;
              WRITELN
            END
       END
```

```
(50)       ELSE
(51)          WRITELN (PAGE [I])
(52) END.
```

<u>הסבר:</u>

14-18 קריאת הקלט עד שורה המתחילה בנקודה.
19-47 הדפסת התמליל כאשר:
20 בדיקה אם השורה מתחילה בתו '#'.
26-31 חישוב מספר הרווחים מהספרות שלאחר התו '#'.
32-33 בדיקה אם קיים מספר, אם יש אחריו רווח (כנדרש בשאלה) ואם תחום המספר חוקי. אם לפחות אחד מהתנאים לא מתקיים מודפסת השורה כפי שנקראה (שורה 32).
35-37 אחרת - מודפסים הרווחים ואחריהם השורה.

תוכנית 8

```
PROGRAM ORDER_WORDS (INPUT, OUTPUT);
CONST
   WORD_LENGTH = 25;
   MAX_WORDS   = 50;
TYPE
   ORDER     = (FIRST, SECOND);
   TEXT_WORD = PACKED ARRAY [1..WORD_LENGTH] OF CHAR;
   WORD_LIST = ARRAY [1..MAX_WORDS, ORDER] OF TEXT_WORD;
VAR
   TWO_WORDS  : TEXT_WORD;
   WORD       : ARRAY [ORDER] OF TEXT_WORD;
   READ_ORDER : ORDER;
   LIST       : WORD_LIST;
   COUNTER, INDEX, LOC : INTEGER;
BEGIN
   COUNTER := 0;
   READLN (TWO_WORDS);
   WHILE  TWO_WORDS [1] <> '.'  DO
    BEGIN
      COUNTER := COUNTER + 1;
      INDEX := 1;
      FOR  READ_ORDER := FIRST  TO  SECOND  DO
       BEGIN
         LOC := 1;
         WHILE  TWO_WORDS [INDEX] = ' '  DO
            INDEX := INDEX + 1;
         REPEAT
            WORD [READ_ORDER][LOC] := TWO_WORDS [INDEX];
            LOC := LOC + 1;
            INDEX := INDEX + 1
         UNTIL  TWO_WORDS [INDEX] = ' ';
```

```
(32)           WORD [READ_ORDER][LOC] := ' '
(33)         END;    { for read_order }
(34)        IF  WORD [FIRST] <= WORD [SECOND]  THEN
(35)         BEGIN
(36)           LIST [COUNTER, FIRST]  := WORD [FIRST];
(37)           LIST [COUNTER, SECOND] := WORD [SECOND]
(38)         END
(39)        ELSE
(40)         BEGIN
(41)           LIST [COUNTER, FIRST]  := WORD [SECOND];
(42)           LIST [COUNTER, SECOND] := WORD [FIRST]
(43)         END;
(44)        READLN (TWO_WORDS)
(45)      END;  { while two_words }
(46)     FOR  INDEX := 1  TO  COUNTER  DO
(47)      BEGIN
(48)        WRITE (INDEX);
(49)        FOR  READ_ORDER := FIRST  TO  SECOND  DO
(50)         BEGIN
(51)           WRITE (' ':5);
(52)           LOC:=1;
(53)           WHILE  LIST [INDEX, READ_ORDER][LOC] <> ' '  DO
(54)            BEGIN
(55)              WRITE (LIST [INDEX, READ_ORDER][LOC]:1);
(56)              LOC := LOC + 1
(57)            END
(58)         END;
(59)        WRITELN
(60)      END  { for index }
(61) END.
```

__הערה:__

התוכנית מתחשבת בסוגי האותיות (קטנות או גדולות), אשר יש להן ערך מספרי שונה. הערך של האותיות הגדולות שונה מהערך של האותיות הקטנות וקטן מהן, כלומר הערך של האות A קטן מהערך של האות a.

למשל: ABC < ABD
ABC < aBC
ABC < AaC   אבל AbC > AaC

__הסבר:__

22-33 הפרדת שתי המלים אשר נקראו במחרוזת אחת והוספה של רווח בסוף כל מלה.

לדוגמה, נתון : TWO_WORDS = '^ABCD^^^EFAC^^'
נקבל : WORD[SECOND]='EFAC^' ; WORD[FIRST]='ABCD^'

34-43 שתי המלים מוחסנות במערך LIST לפי תוצאת ההשואה שלהן.
46-60 הדפסה של רשימת המלים לפי סדר הקלט ולפי סדר המיון של זוגות המלים.

הסבר למשתנים:

INDEX מצביע על מיקום התו הנוכחי במערך שבו נמצאות שתי המלים יחד.

LOC מצביע על המיקום הנוכחי במערך של שתי המלים הנפרדות.

תוכנית 9

```
PROGRAM TEXT_9 (INPUT, OUTPUT);
CONST
   MAX_LENGTH = 80;
TYPE
   TEXT_LINE = PACKED ARRAY [1..MAX_LENGTH] OF CHAR;
VAR
   SENTENCE      : TEXT_LINE;
   INDEX,
   REVACH_RISHON,
   WORD_COUNT    : INTEGER;
   MILA_ACHRONA  : CHAR;
BEGIN
   READLN (SENTENCE);
   INDEX := 1;
   WORD_COUNT := 0;
   REVACH_RISHON := 0;
   MILA_ACHRONA := SENTENCE [1];
   WHILE  SENTENCE [INDEX] <> '.'  DO
    BEGIN
      IF (SENTENCE [INDEX] = ' ') THEN
       BEGIN
        IF (REVACH_RISHON = 0) THEN
           REVACH_RISHON := INDEX;
        IF (SENTENCE [INDEX+1] <> ' ') AND
           (SENTENCE [INDEX+1] <> '.') THEN
           MILA_ACHRONA := SENTENCE [INDEX+1]
       END
      ELSE
        IF (SENTENCE [INDEX+1] = ' ') OR
           (SENTENCE [INDEX+1] = '.')   THEN
             WORD_COUNT := WORD_COUNT + 1;
       INDEX := INDEX + 1
    END   { while sentence }
END.
```

<u>הסבר:</u>

20 מזהה תו רווח במשפט.

22-23 המשתנה REVACH_RISHON מקבל ערך התחלתי 0 ומעודכן רק פעם אחת במהלך התוכנית, כאשר מגיעים לרווח ראשון.

24-26 אם התו שאחרי הרווח איננו רווח או נקודה, אזי המשתנה MILA_ACHRONA מתעדכן לתו זה, שהוא התחלת המלה הבאה במשפט. כאשר המשפט מסתיים, נשמר במשתנה זה התו הראשון של המלה האחרונה.

27-29 כאשר התו הנוכחי איננו רווח ולעומת זאת התו הבא הינו רווח או נקודה - הסתיימה מלה ולכן נוסיף 1 למשתנה WORD_COUNTER.

תוכנית 11

```
PROGRAM TEXT_11 (INPUT, OUTPUT);
VAR
   LETTERS  : ARRAY [1..26] OF INTEGER;
   CH       : CHAR;
   INDEX    : INTEGER;
   NO_ERROR : BOOLEAN;
BEGIN
   FOR  INDEX := 1  TO  26  DO
      LETTERS [INDEX] := 0;
   READ (CH);
   WHILE  CH <> '.'  DO
    BEGIN
      NO_ERROR := TRUE;
      IF (CH >= 'A') AND (CH <= 'Z') THEN
         INDEX := ORD (CH) - ORD ('A') + 1
      ELSE
         IF (CH >= 'a') AND (CH <= 'z') THEN
            INDEX := ORD (CH) - ORD ('a') + 1
         ELSE
            NO_ERROR := FALSE;
      IF NO_ERROR THEN
         LETTERS [INDEX] := LETTERS [INDEX] + 1;
      READ (CH)
    END
END.
```

<u>הסבר:</u>

המשתנה NO_ERROR בודק שאין בתמליל תווים שאינם אותיות.

תוכנית 12

```
PROGRAM TEXT_NUMBERS (INPUT, OUTPUT);
CONST
   NUM_LENGTH = 30;
VAR
   LINE     : PACKED ARRAY [1..NUM_LENGTH] OF CHAR;
   SUM,
   LOOP,
   SIGN,
   NUMBER,
   INDEX    : INTEGER;
   NO_ERROR : BOOLEAN;
BEGIN
   READLN (LINE);
   SUM := 0;
   INDEX := 1;
   NO_ERROR := TRUE;
   FOR  LOOP := 1  TO  10  DO
    BEGIN
      SIGN := 1;
      IF  LINE [INDEX] = ' '  THEN
         INDEX := INDEX + 1;
      IF (LINE [INDEX] = ' ') OR (LINE [INDEX] = '+')
         OR (LINE [INDEX] = '-')  THEN
       BEGIN
         IF  LINE [INDEX] = '-'  THEN
            SIGN := -1;
         INDEX := INDEX + 1
       END;
      NUMBER := 0;
      WHILE  INDEX <= LOOP * 3  DO
       BEGIN
         IF (LINE [INDEX] > '9') OR (LINE [INDEX] < '0') THEN
          BEGIN
            NO_ERROR := FALSE;
            INDEX := LOOP * 3 + 1
          END
         ELSE
          BEGIN
            NUMBER := NUMBER * 10 +
                           ORD (LINE [INDEX]) - ORD ('0');
            INDEX  := INDEX + 1
          END
       END;
      SUM := SUM + NUMBER * SIGN
    END;
   IF NO_ERROR THEN
      WRITELN (SUM, ' הסכום הוא')
   ELSE
      WRITELN ('טעות בקלט!')
END.
```

חסבר:

19	הערך ההתחלתי של משתנה הסימן (SIGN) הוא חיובי.
20-21	דילוג על רווח משמאל (אם יש).
22-28	אם יש משמאל רווח או סימן מעדכנים את הסימן בהתאם.
29-43	בדיקה אם אין תווים שונים מספרות, וחישוב המספר עד המיקום המתאים (המספר N נמצא במערך ממקום 3*N-2 עד מקום 3*N. אם יש טעות בקלט, המשתנה NO_ERROR מקבל FALSE.
44	אחרת המספר שחושב מחובר לסכום הכולל.

תוכנית 13

```
PROGRAM TEXT_13 (INPUT, OUTPUT);
CONST
   MAX_LINE_LENGTH = 70;
   MAX_LINES       = 200;
TYPE
   TEXT_LINE = PACKED ARRAY [1..MAX_LINE_LENGTH] OF CHAR;
   TEXT_PAGE = ARRAY [1..MAX_LINES] OF TEXT_LINE;
VAR
   LINE : TEXT_LINE;
   PAGE : TEXT_PAGE;
   INDEX, LINES, WORDS, SENTENCES, I : INTEGER;
BEGIN
   LINES := 0;
   REPEAT
      LINES := LINES + 1;
      READLN (PAGE [LINES])
   UNTIL  PAGE [LINES, 1] = '.';
   LINES := LINES - 1;       { אין סופרים את השורה המסמנת סוף קלט }
   WORDS := 0;
   SENTENCES := 0;
   FOR  I := 1  TO  LINES  DO
    BEGIN
      FOR  INDEX := 2  TO  MAX_LINE_LENGTH  DO
       BEGIN
         IF (PAGE [I, INDEX] = ' ') AND
            (PAGE [I, INDEX-1] <> ' ')  THEN              { סוף מלה }
            WORDS := WORDS + 1;
         IF (PAGE [I, INDEX] = ' ') AND (PAGE [I, INDEX-1] = '.')
          THEN                                            { סוף משפט }
            SENTENCES := SENTENCES + 1
       END;
      WRITELN (PAGE [I])
    END;
   WRITELN (LINES:50, ' מספר השורות');
```

```
(35)    WRITELN (SENTENCES:49, ' מספר המשפטים');
(36)    WRITELN (WORDS:50, ' מספר המלים')
(37) END.
```

הנחות: * מכסימום 200 שורות.
* אורך שורה מכסימלי 70 תווים.
* סיום התמליל כאשר בתחילת השורה מופיעה נקודה (שורה 17).

תוכנית 14

```
PROGRAM STRINGS (INPUT, OUTPUT);
CONST
   LENGTH = 10;
TYPE
   CHR_STRING = PACKED ARRAY [1..LENGTH] OF CHAR;
VAR
   LIST                                       : CHR_STRING;
   N, INDEX, T_COUNTER, LOC, T_LOC, S_LOC,
   S_BEFORE_T, MAX_T, MAX_T_PTR               : INTEGER;
BEGIN
   READLN (N);
   S_BEFORE_T := 0;
   MAX_T      := 0;
   MAX_T_PTR  := 1;
   FOR  INDEX := 1  TO  N  DO
    BEGIN
      READLN (LIST);
      T_COUNTER := 0;
      T_LOC     := 0;
      S_LOC     := 0;
      FOR  LOC := 1  TO  10  DO
         IF  LIST [LOC] = 'T'  THEN
          BEGIN
            T_COUNTER := T_COUNTER + 1;
            IF  T_LOC = 0  THEN
               T_LOC := LOC
          END
         ELSE
            IF  LIST [LOC] = 'S'  THEN
               IF  S_LOC = 0  THEN
                  S_LOC := LOC;
      IF (S_LOC > 0) AND (S_LOC < T_LOC) THEN
         S_BEFORE_T := S_BEFORE_T + 1;
      IF  T_COUNTER > MAX_T  THEN
       BEGIN
         MAX_T     := T_COUNTER;
         MAX_T_PTR := INDEX
       END
```

```
(39)      END;    { for index... }
(40)     WRITELN ('האות T מופיעה במספר המירבי של פעמים', MAX_T_PTR:1,
(41)              'במחרוזת ה-');
(42)     WRITELN ('מחרוזות':18, S_BEFORE_T:1,
(43)              'האות S מופיעה לפני האות T ב-')
(44) END.
```

<u>הסבר:</u>

22-27 סופרים את מספר הפעמים שהאות T מופיעה במחרוזת. המשתנה T_LOC מקבל את מיקום האות הראשונה במחרוזת.

29-31 מציאת המיקום הראשון של האות S במחרוזת (אם יש).

32-33 <u>אם</u> S מופיעה לפני T, <u>אז</u> הוסף 1 למונה S_BEFORE_T.

34-38 <u>אם</u> מספר האותיות T במחרוזת הנוכחית גדול מהמכסימום עד עכשיו, <u>אז</u> עדכן את מספר המחרוזת הזו ואת מספר המופעים המכסימלי של האות T.

תוכנית 15

```
PROGRAM TEXT_LETTERS (INPUT, OUTPUT);
CONST
   MAX_LINE_LENGTH = 70;
   MAX_LINES       = 100;
   MAX_STR_LENGTH  = 6;
   MAX_NAMES       = 30;
TYPE
   TEXT_LINE   = PACKED ARRAY [1..MAX_LINE_LENGTH] OF CHAR;
   LETTERS     = ARRAY [1..MAX_LINES] OF TEXT_LINE;
   REPLACE_STR = PACKED ARRAY [1..MAX_STR_LENGTH] OF CHAR;
VAR
   LINE        : TEXT_LINE;
   LETTER      : LETTERS;
   NAME, DATE, TEST_STR : REPLACE_STR;
   NAMES_AND_DATES   : ARRAY [1..MAX_NAMES,1..2] OF REPLACE_STR;
   INDEX, I, J, LINES, LETTER_LINES   : INTEGER;
BEGIN
   NAME     := '*NAME*';
   DATE     := '*DATE*';
   TEST_STR := '      ';
   LINES    := 0;
   REPEAT
      LINES := LINES + 1;
      READLN (LETTER [LINES])
   UNTIL  LETTER [LINES, 1] = '.';
   LINES := 0;
   REPEAT
      LINES := LINES + 1;
      READLN (LINE);
      INDEX := 1;
```

```
      IF  LINE [1] <> '.'  THEN
         FOR  I := 1  TO  2  DO
          BEGIN
            WHILE  LINE [INDEX] = ' '  DO
               INDEX := INDEX + 1;
            J := 1;
            REPEAT
               NAMES_AND_DATES [LINES, I, J] := LINE [INDEX];
               J := J + 1;
               INDEX := INDEX + 1
            UNTIL  LINE [INDEX] = ' ';
            NAMES_AND_DATES [LINES, I, J] := ' '
          END
   UNTIL  LINE [1] = '.';
   FOR  INDEX := 1  TO  LINES - 1  DO
    BEGIN
      LETTER_LINES := 1;
      WHILE  LETTER [LETTER_LINES, 1] <> '.'  DO
       BEGIN
         I := 1;
         WHILE  I < MAX_LINE_LENGTH - 5  DO
          BEGIN
           IF  LETTER [LETTER_LINES, I] = '*'  THEN
            BEGIN
             FOR  J := 1  TO  6  DO
               TEST_STR [J] := LETTER [LETTER_LINES, I + J - 1];
             IF  TEST_STR = NAME  THEN
              BEGIN
               I := I + 6;
               J := 1;
               WHILE  NAMES_AND_DATES [INDEX, 1, J] <> ' '  DO
                BEGIN
                 WRITE (NAMES_AND_DATES [INDEX, 1, J]:1);
                 J := J + 1
                END
              END
             ELSE
               IF  TEST_STR = DATE  THEN
                BEGIN
                 I := I + 6;
                 J := 1;
                 WHILE  NAMES_AND_DATES [INDEX, 2, J] <> ' ' DO
                  BEGIN
                   WRITE (NAMES_AND_DATES [INDEX, 2, J]:1);
                   J := J + 1
                  END
                END
               ELSE
                BEGIN
                 WRITE (TEST_STR [1]:1);
                 I := I + 1
```

```
(82)                END
(83)            END    { if letter ... }
(84)           ELSE
(85)            BEGIN
(86)             WRITE (LETTER [LETTER_LINES, I]:1);
(87)             I := I + 1
(88)            END
(89)          END;  { while i<max_line..  }
(90)         WRITELN;
(91)         LETTER_LINES := LETTER_LINES + 1
(92)       END;   { while letter... }
(93)      WRITELN;
(94)      WRITELN
(95)    END
(96) END.
```

<u>הסבר:</u>

21-25 קריאת המכתב למערך.

26-44 קריאת רשימת השמות והתאריכים והפרדתם, תוך כדי החסנה במערך NAMES_AND_DATES.
המכתב ורשימת השמות מסתיימים בשורה המתחילה בנקודה.

45-95 סריקת המכתב בלולאה עד למציאת המחרוזות *NAME* ו-*DATE*. כאשר נתקלים ב-'*' משווים אותו ואת 5 התווים שאחריו למחרוזות *NAME* ו- *DATE*. <u>אם</u> ישנה התאמה, <u>אז</u> מדפיסים את המחרוזת המתאימה מ-NAMES_AND_DATES.

תוכנית 16

```
PROGRAM TEXT_LINES (INPUT, OUTPUT);
CONST
   MAX_LINE_LENGTH  = 80;
   MAX_LINES        = 100;
   MAX_TEXT_LENGTH  = 8000;
   EDIT_LINE_LENGTH = 60;
TYPE
   ALL_TEXT  = PACKED ARRAY [1..MAX_TEXT_LENGTH] OF CHAR;
   TEXT_LINE = PACKED ARRAY [1..MAX_LINE_LENGTH] OF CHAR;
VAR
   LINE  : TEXT_LINE;
   PAGE  : ALL_TEXT;
   INDEX, CHR_COUNTER, LINE_COUNTER, PRESENT_POSITION : INTEGER;
   NEW_LINE        : BOOLEAN;
BEGIN
   PRESENT_POSITION := 0;            { מצביע על המקום האחרון המלא }
   REPEAT                            { במערך PAGE בקריאה }
      LINE_COUNTER := MAX_LINE_LENGTH;
      READLN (LINE);
```
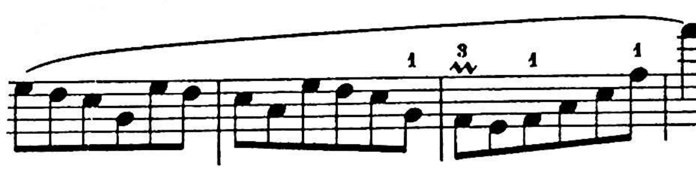
```
(20)       WHILE (LINE_COUNTER > 0) AND (LINE [LINE_COUNTER] = ' ') DO
(21)          LINE_COUNTER := LINE_COUNTER - 1;
(22)       FOR INDEX := 1  TO  LINE_COUNTER  DO
(23)          PAGE [PRESENT_POSITION + INDEX] := LINE [INDEX];
(24)       PRESENT_POSITION := PRESENT_POSITION + LINE_COUNTER + 1;
(25)       PAGE [PRESENT_POSITION] := ' '
(26)    UNTIL  LINE [1] = '.';
(27)    PRESENT_POSITION := PRESENT_POSITION - 2;
(28)    WRITELN;
(29)    CHR_COUNTER := 0;  { מצביע על המקום הנוכחי במערך PAGE בהדפסה }
(30)    NEW_LINE    := FALSE;                 { מצביע על תחילת הפיסקה }
(31)    WHILE CHR_COUNTER < PRESENT_POSITION DO  { תנאי לסיום הסריקה }
(32)     BEGIN
(33)       WHILE  PAGE [CHR_COUNTER + 1] = ' ' DO  { דילוג על רווחים }
(34)         CHR_COUNTER := CHR_COUNTER + 1;     {       בתחילת שורה }
(35)       INDEX := CHR_COUNTER + EDIT_LINE_LENGTH;
(36)       IF NEW_LINE THEN                     { אם זוהי התחלת פיסקה }
(37)        BEGIN
(38)          NEW_LINE := FALSE;
(39)          WRITE (' ':4);             {                 הדפסת 4 רווחים }
(40)          INDEX := INDEX - 4         { הפחתת 4 מאורך השורה שתודפס }
(41)        END;
(42)       IF INDEX >= PRESENT_POSITION THEN { אם מגיעים לסוף התמליל }
(43)                          { לפני סוף שורה אז INDEX יצביע על סוף התמליל. }
(44)         INDEX := PRESENT_POSITION - 1
(45)       ELSE                    { חיפוש כלפי מטה למציאת רווח בין המלים }
(46)         IF (PAGE [INDEX+1] <> ' ') AND (PAGE [INDEX] <> ' ') THEN
(47)            WHILE (PAGE [INDEX] <> ' ') DO
(48)               INDEX := INDEX - 1;
(49)       WHILE  (CHR_COUNTER < INDEX) AND NOT NEW_LINE  DO
(50)        BEGIN                  { הדפסה עד המקום ש-INDEX מצביע עליו, }
(51)                               {                 או עד זיהוי תחילת פיסקה. }
(52)          CHR_COUNTER := CHR_COUNTER + 1;
(53)          IF  PAGE [CHR_COUNTER] = '/'  THEN { סימן לתחילת פיסקה }
(54)             NEW_LINE := TRUE
(55)          ELSE
(56)             WRITE (PAGE [CHR_COUNTER]:1)     { הדפסת התו הנוכחי }
(57)        END;
(58)       WRITELN                                        { שורה חדשה }
(59)     END    { while chr_counter }
(60) END.
```

הסבר:

16-26 קריאת הקלט עד שורה שמתחילה בנקודה.
כל שורת קלט שנקראת למערך LINE עוברת סריקה מהסוף להתחלה עד למציאת התו האחרון. השורה מועתקת למערך PAGE לאחר השורה הקודמת ומפרידים ביניהן ברווח.
המשתנה PRESENT_POSITION מצביע על המקום האחרון המלא במערך PAGE. במערך PAGE מתקבלת שורה אחת ארוכה של תמליל.

27 הורדת הנקודה והרווח המסמנים את סוף התמליל.
עוברים על מערך PAGE. בכל פעם מתחילים משורה של 60 תווים
(או 56, אם זו תחילת פיסקה (38-43)). בודקים אם עוברים את סוף
התמליל (44-45), או שסוף שורה יגרום לחיתוך מלה (47-49).

52-59 לאחר שחושב אורך השורה הנוכחית להדפסה, בודקים כל תו בחיפוש
אחר '/'. כאשר תו שכזה נמצא, מפסיקים את ההדפסה ומתחילים פיסקה
חדשה.



"מחרוזת תוים"

פרק 7
חיפוש, מיון ומיזוג

שאלות

שאלה 2

```
PROGRAM INSERTION;
CONST
   N = 50;
TYPE
   VALUE      = INTEGER;
   SORT_ARRAY = ARRAY [1..N] OF VALUE;
VAR
   A         : SORT_ARRAY;
   I, J, K   : INTEGER;
   X         : VALUE;
   FOUND     : BOOLEAN;
BEGIN
   FOR  I := 2  TO  N  DO
    BEGIN
      X := A [I];
      J := 1;
      FOUND := FALSE;                           { חיפוש בחלק }
      WHILE (J < I) AND NOT FOUND DO            { הממויין של המערך. }
         IF  X < A [J]  THEN                    { אחר איבר ש-X קטן ממנו. }
            FOUND := TRUE                       { איבר כזה נמצא }
         ELSE
            J := J + 1;                         { איבר כזה עדיין לא נמצא }
      FOR  K := I - 1  DOWNTO  J  DO
         A [K + 1] := A [K];
      A [J] := X
    END      { for i }
END.
```

הסבר:

משווים את האיבר השני לאיבר הראשון. <u>אם</u> הוא קטן ממנו <u>אז</u> הוא מוכנס לפניו, <u>אחרת</u> - הוא ישאר במקומו.
במעבר הבא משווים את האיבר השלישי לשני האיברים שכבר ממויינים ומכניסים אותו במקום המתאים באותה דרך שהוסברה בספר. ההבדל הוא בכך, שהנתונים לא מתקבלים כקלט בכל שלב, אלא נמצאים במקום הבא במערך שיש למיין.
הסוג VALUE יכול להיות כל סוג משתנה שהיחס > (גדול) מוגדר לגביו. למשל: BOOLEAN, CHAR, REAL וכו'.

שאלה 5

```
PROGRAM BUBBLE;
CONST
   N = 100;
TYPE
   VALUE      = INTEGER;
   SORT_ARRAY = ARRAY [1..N] OF VALUE;
VAR
   A    : SORT_ARRAY;
   I, J, K, L : INTEGER;
   X    : VALUE;
BEGIN
   J := N;
   K := 1;
   WHILE  J > K  DO
    BEGIN
      L := 0;
      FOR  I := K  TO  J - 1  DO
         IF  A [I] > A [I + 1]  THEN
          BEGIN
            X         := A [I];
            A [I]     := A [I + 1];
            A [I + 1] := X;
            L := I                   { מתעדכן כל פעם שמבוצעת החלפה }
          END;
      J := L;
      L := J;                        { מקום החלפה אחרון - עליון }
      FOR  I := J - 1  DOWNTO  K  DO
         IF  A [I] > A [I + 1]  THEN
          BEGIN
            X         := A [I];
            A [I]     := A [I + 1];
            A [I + 1] := X;
            L := I                   { מתעדכן כל פעם שמבוצעת החלפה }
          END;
      K := L                         { מקום החלפה אחרון - תחתון }
    END  { while j > k }
END.
```

הסבר:

17-25 סריקה של הווקטור מהאיבר הראשון אל האיבר האחרון והחלפה של זוגות איברים אם הם אינם ממויינים. המשתנה J שבשורה 25 מצביע על מיקום האיבר האחרון שהוחלף (כל האיברים שאחריו כבר ממויינים).

26-35 סריקה של הווקטור בכיוון הפוך, כלומר מלמעלה (אינדקס גבוה) למטה (אינדקס נמוך). שים לב שהסריקה מתחילה מקום אחד לאחר האיבר האחרון שהוחלף בסריקה הקודמת ומסומן על-ידי J. המשתנה K בשורה 35 מצביע על האיבר האחרון שהוחלף. האיברים שאחריו כבר ממויינים. המיון מסתיים כאשר אין עוד איברים בין J ל-K, כלומר K <= J.

תוכניות

תוכנית 1

```
PROGRAM PAY_SORT (INPUT, OUTPUT);
CONST
   MAX_WORKS = 50;
TYPE
   NAME        = PACKED ARRAY [1..20] OF CHAR;
   NAME_ARRAY  = ARRAY [1..MAX_WORKS] OF NAME;
   VETEK_ARRAY = ARRAY [1..MAX_WORKS] OF INTEGER;
VAR
   V, I, J, N  : INTEGER;
   WORKER_NAME : NAME;
   NAMES       : NAME_ARRAY;
   VETEK       : VETEK_ARRAY;
   FOUND       : BOOLEAN;
   LIMIT       : ARRAY [1..3,0..6] OF INTEGER;
BEGIN
   LIMIT [1, 1] := 3;    {
   LIMIT [1, 2] := 6;    {
   LIMIT [1, 3] := 11;   {
   LIMIT [1, 4] := 14;   {
   LIMIT [1, 5] := 21;   {
   LIMIT [1, 6] := 26;   {
   LIMIT [2, 1] := 0;    {
   LIMIT [2, 2] := 475;  {
   LIMIT [2, 3] := 578;  {
   LIMIT [2, 4] := 682;  {
   LIMIT [2, 5] := 810;
   LIMIT [2, 6] := 950;
   FOR I := 0 TO 6 DO
      LIMIT [3, I] := 0;
   N := 0;
   READLN (V, WORKER_NAME);
```

3	6	11	14	21	26
0	475	578	682	810	950
	↑				

מספר העובדים שתחום הותק שלהם הוא בין 3 ל-(1-6) שנים. משכורתם היא 475

```
    WHILE  V >= 0  DO
     BEGIN
       N := N + 1;
       FOUND := FALSE;
       J := 1;
       WHILE (J < N) AND NOT FOUND DO
          IF  V < VETEK [J]  THEN
             FOUND := TRUE
          ELSE
             J := J + 1;
       FOR  I := N  DOWNTO  J  DO
        BEGIN
          VETEK [I + 1] := VETEK [I];
          NAMES [I + 1] := NAMES [I]
        END;
       VETEK [J] := V;
       NAMES [J] := WORKER_NAME;
       READLN (V, WORKER_NAME)
     END;   { while v }
    J := 1;
    WRITELN ('שם          משכורת מירבית':53);
    WHILE  (J <= N) AND (VETEK [J] < LIMIT [1, 1]) DO
     BEGIN
       WRITELN ('ללא כלל - פחות מ-3 שנות וותק':40, NAMES [J]:25);
       J := J + 1;
       LIMIT [3, 0] := LIMIT [3, 0] + 1
     END;
    I := 1;
    WHILE  (J <= N) AND (VETEK [J] < LIMIT [1, 6]) DO
     BEGIN
       WHILE  VETEK [J] >= LIMIT [1, I] DO
          I := I + 1;
       WRITELN (LIMIT [2, I]:40, NAMES [J]:25);
       J := J + 1;
       LIMIT [3, I-1] := LIMIT [3, I-1] + 1
     END;
    WHILE  J <= N  DO
     BEGIN
       WRITELN ('ללא כלל- יותר מ-25 שנות וותק':40 , NAMES [J]:25);
       J := J + 1;
       LIMIT [3, 6] := LIMIT [3, 6] + 1
     END;
    WRITELN;
    WRITELN ('מספר עובדים          שנות וותק':50);
    WRITELN ('פחות מ-3 שנות וותק':30, LIMIT [3, 0]:15);
    FOR  I := 1  TO  5  DO
      WRITELN (LIMIT [1, I]:27, '-', LIMIT [1, I+1]:2,
                 LIMIT [3, I]:15);
    WRITELN ('יותר מ-25 שנות וותק':30, LIMIT [3, 6]:15)
END.
```

<u>הסבר:</u>

המבנה של המערך LIMIT:
- בשורה הראשונה נמצא התחום התחתון של שנות ותק. התחום העליון הוא הערך של התא הבא פחות אחד (ראה תרשים בגוף התוכנית).
- בשורה השניה נמצאת המשכורת.
- בשורה השלישית נשתמש כמונה למספר העובדים ששנות הותק שלהם נמצאות באותו תחום ותק.

30-50 לולאה של קריאת שמות העובדים ושנות הותק שלהם, תוך כדי בניית מערך ממויין, לפי אלגוריתם של מיון על ידי שיבוץ.
53-58 הדפסה וספירה של העובדים מתחת ל-3 שנות ותק.
59-67 הדפסה וספירה של העובדים מ-3 ועד 25 שנות ותק.
68-73 הדפסה וספירה של העובדים מעל 25 שנות ותק.
74-80 הדפסת טבלה של מספר העובדים בכל תחום שנות ותק.

תוכנית 2

```
PROGRAM HALF (INPUT, OUTPUT);
CONST
   MAX = 100;
TYPE
   VALUE      = INTEGER;
   SORT_ARRAY = ARRAY [1..MAX] OF VALUE;
VAR
   A     : SORT_ARRAY;
   I, J, K, L, N : INTEGER;
   X     : VALUE;
BEGIN
   READLN (N);
   FOR  I := 1  TO  N  DO
      READLN (A [I]);
   J := N;
   K := 1;
   WHILE  J > K  DO
    BEGIN
      L := 0;
      FOR  I := K  TO  J - 1  DO
         IF  A [I] > A [I + 1]  THEN
          BEGIN
            X  := A [I];
            A [I]  := A [I + 1];
            A [I + 1] := X;
            L := I
          END;
      J := L;
      L := J;
```

```
(30)         FOR  I := J - 1  DOWNTO  K  DO
(31)            IF  A [I] > A [I + 1]  THEN
(32)             BEGIN
(33)               X           := A [I];
(34)               A [I]       := A [I + 1];
(35)               A [I + 1] := X;
(36)               L := I
(37)             END;
(38)         K := L
(39)       END;    { while j > k }
(40)     IF  ODD (N)  THEN
(41)        WRITELN (A [(N + 1) DIV 2]:1, 'החציון הוא ')
(42)     ELSE
(43)        WRITELN ((A [N DIV 2] + A [N DIV 2 + 1]) / 2 :4:2,
(44)              'החציון הוא ')
(45) END.
```

<u>הסבר:</u>

15-39 מיון על ידי שיבוץ (סריקה) עם גבול עליון וגבול תחתון דינמיים.
40-44 חישוב החציון לפי הכלל שמוגדר בשאלה.
הפונקציה ODD(X) מחזירה ערך TRUE אם X הוא מספר אי-זוגי.

תוכנית 4

```
PROGRAM MERGING (INPUT, OUTPUT);
CONST
   MAX_NAMES = 300;
TYPE
   NAME  = PACKED ARRAY [1..20] OF CHAR;
   NAMES = ARRAY [1..MAX_NAMES] OF NAME;
VAR
   NEW_NAMES, OLD_NAMES, SORT_NAMES : NAMES;
   I, M, N                          : INTEGER;

PROCEDURE READ_NAMES (VAR  X : NAMES;           {       מערך של שמות }
                      VAR  N : INTEGER);        { מספר השמות שנקראו }
VAR
   I : INTEGER;
BEGIN
   N := -1;
   WRITELN ('הכנס את השמות':15);
   REPEAT
      N := N + 1;
      READLN (X [N + 1])
   UNTIL  X [N + 1, 1] = '.'
END;
```

```
                                                { מיון בועות של }
PROCEDURE SORT_ARRAY (VAR  A : NAMES;           { מערך שמות A }
                           N : INTEGER);        { המכיל N איברים }
VAR
   I, J, L : INTEGER;
   X   : NAME;
BEGIN
   J := N;
   WHILE  J > 0  DO
    BEGIN
      L := 0;
      FOR  I := 1  TO  J - 1  DO
         IF  A [I] > A [I + 1]  THEN
          BEGIN
            X := A [I];
            A [I] := A [I + 1];
            A [I + 1] := X;
            L := I
          END;
      J := L
    END
END;

PROCEDURE MERGE (VAR A, B, C : NAMES;
                         M, N : INTEGER);
VAR
   I, J, K, P : INTEGER;
BEGIN
   I := 1;
   J := 1;
   K := 0;
   WHILE (I <= M) AND (J <= N) DO
    BEGIN
      K := K + 1;
      IF  A [I] < B [J]  THEN
       BEGIN
         C [K] := A [I];
         I := I + 1
       END
      ELSE
       BEGIN
         C [K] := B [J];
         IF  A [I] = B [J]  THEN
            I := I + 1;
         J := J + 1
       END
    END;
   IF  I <= M  THEN
      FOR  P := I  TO  M  DO
       BEGIN
         K := K + 1;
```

```
(71)           C [K] := A [P]
(72)         END
(73)       ELSE
(74)          IF  J <= N  THEN
(75)             FOR  P := J  TO  N  DO
(76)              BEGIN
(77)                K := K + 1;
(78)                C [K] := B [P]
(79)              END
(80) END;

(81) BEGIN  { main }
(82)    WRITELN ('רשימת התלמידים החדשים':50);
(83)    READ_NAMES (NEW_NAMES, M);
(84)    WRITELN ('רשימת התלמידים הוותיקים':50);
(85)    READ_NAMES (OLD_NAMES, N);
(86)    SORT_ARRAY (NEW_NAMES, M);
(87)    SORT_ARRAY (OLD_NAMES, N);
(88)    MERGE (NEW_NAMES, OLD_NAMES, SORT_NAMES, M, N);
(89)    WRITELN ('רשימת התלמידים הממויינת':50);
(90)    FOR  I := 1  TO  N + M  DO
(91)       WRITELN (SORT_NAMES [I])
(92) END.
```

<u>הסבר:</u>

בעיה זו ניתן לפתור בשתי דרכים:

א. על-ידי חיבור שני הוקטורים לוקטור אחד, וביצוע מיון אחד על כל השמות.

ב. על ידי ביצוע מיזוג.

הפתרון המוצג בשאלה זו הינו מיון של הוקטורים ע"י מיזוג (התנאי למיזוג הוא, שהוקטורים יהיו ממויינים. קטע המיזוג (43-80) מופיע בספר פסקל בעמודים 162-163).

הפרוצדורה READ_NAMES קוראת את שמות התלמידים עד שמופיעה נקודה במקום של התו הראשון בשם הנקרא, כי זהו סימן לסוף הקלט. הפרוצדורה מחזירה מערך של שמות ואת מספר השמות שהתקבלו. בפרוצדורה זו משתמשים פעמיים: בשורה 83 לקריאת שמות התלמידים החדשים, ובשורה 85 לקריאת השמות של התלמידים הוותיקים.

הפרוצדורה SORT_ARRAY בשורות 42-22 מקבלת מערך של איברים ואת מספר האיברים שבו, ומחזירה אותו ממויין (מיון בועות). בפרוצדורה זו נשתמש בשורות 86 ו-87 למיון הוקטורים המכילים את שמות התלמידים החדשים והוותיקים.

הפרוצדורה MERGE מקבלת 2 מערכים ממויינים A ו-B, ואת מספר האיברים בכל מערך, N ו-M בהתאמה. הפרוצדורה מחזירה מערך שלישי C ממויין, המכיל את השמות משני המערכים (אלגוריתם של מיון על ידי מיזוג). שורה 88 מפעילה פרוצדורה זו.

89-91 הדפסת הרשימה הממויינת.

תוכנית 5

```
PROGRAM SEARCH (INPUT, OUTPUT);
CONST
   MAX_LOST = 100;
TYPE
   NUMBER      = PACKED ARRAY [1..9] OF CHAR;
   STR         = PACKED ARRAY [1..30] OF CHAR;
   NAME        = PACKED ARRAY [1..20] OF CHAR;
   TELEPHONE   = ARRAY [1..MAX_LOST] OF NUMBER;
   NAMES       = ARRAY [1..MAX_LOST] OF NAME;
   DESCRIPTION = ARRAY [1..MAX_LOST] OF STR;
VAR
   FOUND     : DESCRIPTION;
   FINDER    : NAMES;
   TL_NUMBER : TELEPHONE;
   LOSER, X  : NAME;
   DESC      : STR;
   NUM       : NUMBER;
   LOST      : STR;
   I, J, L, N, FIRST, LAST, MID : INTEGER;
BEGIN
   WRITELN ('מספר המציאות?':15);
   READLN (N);
   FOR  I := 1  TO  N  DO
    BEGIN
      WRITELN ('תאור המציאה:':15);
      READLN (FOUND [I]);
      WRITELN ('שם המוצא:':15);
      READLN (FINDER [I]);
      WRITELN ('מספר טלפון:':15);
      READLN (TL_NUMBER [I])
    END;
   J := N;
   WHILE  J > 0  DO
    BEGIN
      L := 0;
      FOR  I := 1  TO  J - 1  DO
         IF  FOUND [I] > FOUND [I + 1]  THEN
          BEGIN
            DESC := FOUND [I];
            FOUND [I] := FOUND [I + 1];
            FOUND [I + 1] := DESC;
            X := FINDER [I];
            FINDER [I] := FINDER [I + 1];
            FINDER [I + 1] := X;
            NUM := TL_NUMBER [I];
            TL_NUMBER [I] := TL_NUMBER [I + 1];
            TL_NUMBER [I + 1] := NUM;
            L := I
          END;
```

```
(50)        J := L
(51)      END;    { while j > 0 }
(52)     WRITELN ('תאור האבידה:':15);
(53)     READLN (LOST);
(54)     WRITELN ('שם המאבד:':15);
(55)     READLN (LOSER);
(56)     FIRST := 1;
(57)     LAST  := N;
(58)     REPEAT
(59)        MID := (FIRST + LAST) DIV 2;
(60)        IF  LOST < FOUND [MID] THEN
(61)           LAST := MID - 1
(62)        ELSE
(63)           FIRST := MID + 1
(64)     UNTIL  (LOST = FOUND [MID]) OR (FIRST > LAST);
(65)     IF  LOST = FOUND [MID]  THEN
(66)      BEGIN
(67)        WRITELN ('מס. טלפון':9, 'שם המוצא':15, 'שם המאבד':20,
(68)                 'תיאור המציאה':25);
(69)        WRITELN (TL_NUMBER [MID]:9, FINDER [MID]:20, LOSER:20,
(70)           FOUND [MID]:30)
(71)      END
(72) END.
```

הסבר:

21-31 קריאה של רשימת המציאות.
32-51 מיון הרשימה בעזרת האלגוריתם של מיון בועות (ניתן גם לבצע מיון על ידי שיבוץ בתוך מהלך קריאת הנתונים).
52-55 נתוני האבידה.
56-64 חיפוש בינרי של האבידה ברשימה הממויינת (אלגוריתם חיפוש בינרי).

תוכנית 7

```
PROGRAM MARK_SORT (INPUT, OUTPUT);
CONST
   MAX_STUDENT  = 100;
   MAX_QUESTION = 20;
TYPE
   NAME = PACKED ARRAY [1..20] OF CHAR;
VAR
   TZIYUN  : ARRAY [1..MAX_STUDENT, 1..MAX_QUESTION] OF INTEGER;
   MISHKAL : ARRAY [1..MAX_QUESTION] OF INTEGER;
   NAMES   : ARRAY [1..MAX_STUDENT] OF NAME;
   MARK    : ARRAY [1..MAX_STUDENT] OF REAL;
   N, L, I, J, X, M : INTEGER;
   TEMP_MARK  : REAL;
   TEMP_NAME  : NAME;
```

```
BEGIN
   WRITELN ('מספר הסטודנטים');
   READLN (N);
   WRITELN ('מספר השאלות');
   READLN (M);
   FOR  I := 1  TO  N  DO
    BEGIN
      WRITELN ('שם הסטודנט');
      READLN (NAMES [I]);
      WRITELN ('רשימת הציונים');
      FOR  J := 1  TO  M  DO
         READ (TZIYUN [I, J]);
      WRITELN
    END;   { for i }
   X := 0;
   FOR  I := 1  TO  M  DO
    BEGIN
      WRITELN (I:1, 'משקל שאלה מספר');
      READLN (MISHKAL [I]);
      X := X + MISHKAL [I]                         { סכום משקלי הציונים }
    END;
   FOR  I := 1  TO  N  DO
    BEGIN
      MARK [I] := 0;
      FOR  J := 1  TO  M  DO              { סכום המכפלה: ציון * משקל }
         MARK [I] := MARK [I] + MISHKAL [J] * TZIYUN [I, J];
      MARK [I] := MARK [I] / X                     { חישוב ממוצע משוקלל }
    END;
   J := N;
   WHILE  J > 0  DO
    BEGIN
      L := 0;
      FOR  I := 1  TO  J - 1  DO
         IF  MARK [I] < MARK [I + 1]  THEN
          BEGIN
            TEMP_MARK    := MARK [I];
            MARK [I]     := MARK [I + 1];
            MARK [I + 1] := TEMP_MARK;
            TEMP_NAME     := NAMES [I];
            NAMES [I]     := NAMES [I + 1];
            NAMES [I + 1] := TEMP_NAME;
            L := I
          END;
      J := L
    END;   { while j > 0 }
   WRITELN ('שם תלמיד                ציון':35);
   FOR  I := 1  TO  N  DO
      WRITELN (NAMES [I]:25, MARK [I]:10:2)
END.
```

```
(50)        J := L
(51)      END;   { while j > 0 }
(52)     WRITELN ('תאור האבידה:':15);
(53)     READLN (LOST);
(54)     WRITELN ('שם המאבד:':15);
(55)     READLN (LOSER);
(56)     FIRST := 1;
(57)     LAST  := N;
(58)     REPEAT
(59)        MID := (FIRST + LAST) DIV 2;
(60)        IF  LOST < FOUND [MID] THEN
(61)           LAST := MID - 1
(62)        ELSE
(63)           FIRST := MID + 1
(64)     UNTIL  (LOST = FOUND [MID]) OR (FIRST > LAST);
(65)     IF  LOST = FOUND [MID]  THEN
(66)      BEGIN
(67)        WRITELN ('מס. טלפון':9, 'שם המוצא':15, 'שם המאבד':20,
(68)                 'תיאור המציאה':25);
(69)        WRITELN (TL_NUMBER [MID]:9, FINDER [MID]:20, LOSER:20,
(70)           FOUND [MID]:30)
(71)      END
(72) END.
```

<u>הסבר:</u>

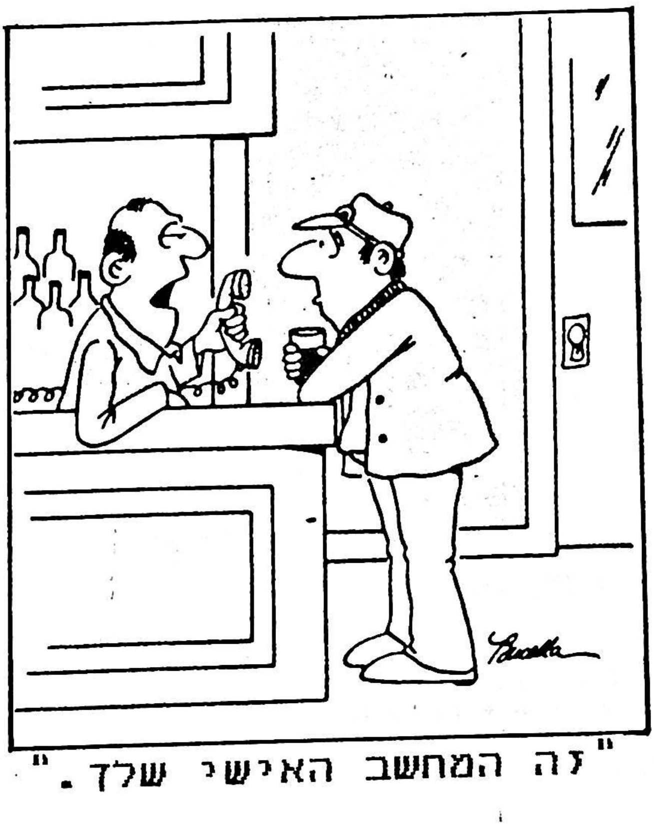
21-31 קריאה של רשימת המציאות.
32-51 מיון הרשימה בעזרת האלגוריתם של מיון בועות (ניתן גם לבצע מיון על ידי שיבוץ בתוך מהלך קריאת הנתונים).
52-55 נתוני האבידה.
56-64 חיפוש בינרי של האבידה ברשימה הממויינת (אלגוריתם חיפוש בינרי).

תוכנית 7

```
PROGRAM MARK_SORT (INPUT, OUTPUT);
CONST
   MAX_STUDENT  = 100;
   MAX_QUESTION = 20;
TYPE
   NAME = PACKED ARRAY [1..20] OF CHAR;
VAR
   TZIYUN  : ARRAY [1..MAX_STUDENT, 1..MAX_QUESTION] OF INTEGER;
   MISHKAL : ARRAY [1..MAX_QUESTION] OF INTEGER;
   NAMES   : ARRAY [1..MAX_STUDENT] OF NAME;
   MARK    : ARRAY [1..MAX_STUDENT] OF REAL;
   N, L, I, J, X, M : INTEGER;
   TEMP_MARK  : REAL;
   TEMP_NAME  : NAME;
```

הסבר:

16-35 קריאת נתונים.
36-42 שיקלול הציון הסופי.
43-59 מיון בועות לפי הציון הסופי.
60-62 הדפסת הרשימה הממויינת.



"זה המחשב האישי שלך."

תוכניות מיון QuickSort ו-HeapSort

QuickSort

```
CONST
   MAX = 512;
TYPE
   ELEMENT_TYPE = INTEGER;
   INDEX = 0..MAX;
   SORT_ARRAY = ARRAY [INDEX] OF ELEMENT_TYPE;

PROCEDURE SWAPP (VAR  X,                { פרוצדורה להחלפת איברים X ו-Y }
                      Y  : ELEMENT_TYPE);
   VAR
      TEMP : ELEMENT_TYPE;
   BEGIN  { swapp }
      TEMP := X;
      X := Y;
      Y := TEMP
   END;   { swapp }

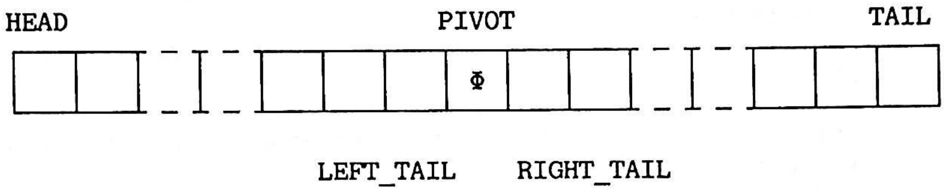
PROCEDURE PARTITION (VAR A     : SORT_ARRAY;
                         HEAD,
                         TAIL : INDEX;
                     VAR  LEFT_TAIL,
                          RIGHT_HEAD : INDEX);
   VAR
      PIVOT : ELEMENT_TYPE;
      L,
      R : INDEX;
   BEGIN  { partition }
      PIVOT := A [(HEAD + TAIL) DIV 2];                { בחירת איבר ציר }
      L := HEAD - 1;
      R := TAIL + 1;
```

```
    REPEAT
       REPEAT
          L := L + 1
       UNTIL  A [L] >= PIVOT;       { לולאה עד איבר הגדול מאיבר הציר }
       REPEAT
          R := R - 1
       UNTIL  A [R] <= PIVOT;       { לולאה עד איבר הקטן מאיבר הציר }
       IF  L < R  THEN                                   { החלף איברים }
          SWAPP (A [L], A [R])
    UNTIL  L >= R;                    { לולאה עד שהמצביע השמאלי גדול
                                     או שווה למצביע הימני }
    IF  L = R  THEN                              { אם המצביעים שווים }
    BEGIN
       LEFT_TAIL := L - 1;
       RIGHT_HEAD := L + 1
    END                                                  { אם L > R }
    ELSE
    BEGIN
       LEFT_TAIL := R;
       RIGHT_HEAD := L
    END
 END;  { partition }

PROCEDURE QUICKSORT (VAR A     : SORT_ARRAY;
                         HEAD,
                         TAIL  : INDEX);

 VAR
    LEFT_TAIL,
    RIGHT_HEAD : INDEX;
 BEGIN { quicksort }
    IF  HEAD < TAIL  THEN
    BEGIN
       PARTITION (A, HEAD, TAIL, LEFT_TAIL, RIGHT_HEAD);  { חלוקת
                                                 המערך לשני חלקים. }
       QUICKSORT (A, HEAD, LEFT_TAIL);              { מיון רקורסיבי
                                                 על החלק השמאלי. }
       QUICKSORT (A, RIGHT_HEAD, TAIL)              { מיון רקורסיבי
                                                  על החלק הימני. }
    END
 END;  { quicksort }
```

הסבר:



A הוא מערך של אלמנטים שיש למיינם לפי סדר.
המשתנים HEAD ו-TAIL קובעים תת-תחום במערך שאותו יש למיין.

אופן הפעולה:

- בחירת אלמנט מהמערך בין HEAD ל-TAIL, במקרה שלנו בוחרים איבר ממרכז המערך (אם כי ניתן לבחור איבר כלשהוא אחר לדוגמא האיבר הראשון במערך) וזהו משתנה PIVOT.

- ארגון האיברים מחדש בצורה שהערך שנבחר (PIVOT) יהיה במקום ה-J ויתקיים:

1. כל האיברים מ-HEAD עד J-1 קטנים או שווים לאיבר במקום ה-J.
2. כל האיברים מ-J+1 ועד TAIL גדולים או שווים לאיבר במקום ה-J.

שים לב שמיקומו של האיבר הנבחר (לאחר ביצוע PARTITION), J, תלוי בערך האיבר הנבחר ומיקום זה יהיה גם מיקומו במערך הממויין.

תהליך זה חוזר בצורה רקורסיבית לגבי תת-מערכים HEAD עד J-1, ו-J+1 עד TAIL.

דוגמה מספרית למיון QUICK SORT:

ניקח את הסדרה הבאה:

מקרא:

H - HEADER
T - TAIL
P - PIVOT
L - LEFT_TAIL
R - RIGHT_TAIL

```
   REPEAT
      REPEAT
         L := L + 1
      UNTIL  A [L] >= PIVOT;      { לולאה עד איבר הגדול מאיבר הציר }
      REPEAT
         R := R - 1
      UNTIL  A [R] <= PIVOT;      { לולאה עד איבר הקטן מאיבר הציר }
      IF  L < R  THEN                                    { החלף איברים }
         SWAPP (A [L], A [R])
   UNTIL  L >= R;                  { לולאה עד שהמצביע השמאלי גדול
            { או שווה למצביע הימני }
   IF  L = R  THEN                                 { אם המצביעים שווים }
   BEGIN
      LEFT_TAIL := L - 1;
      RIGHT_HEAD := L + 1
   END                                                      { אם L > R }
   ELSE
   BEGIN
      LEFT_TAIL := R;
      RIGHT_HEAD := L
   END
  END;  { partition }

PROCEDURE QUICKSORT (VAR A     : SORT_ARRAY;
                         HEAD,
                         TAIL  : INDEX);

  VAR
     LEFT_TAIL,
     RIGHT_HEAD : INDEX;
  BEGIN { quicksort }
     IF  HEAD < TAIL  THEN
     BEGIN
        PARTITION (A, HEAD, TAIL, LEFT_TAIL, RIGHT_HEAD);   { חלוקת }
                                                { המערך לשני חלקים. }
        QUICKSORT (A, HEAD, LEFT_TAIL);              { מיון רקורסיבי }
                                                 { על החלק השמאלי. }
        QUICKSORT (A, RIGHT_HEAD, TAIL)              { מיון רקורסיבי }
                                                  { על החלק הימני. }
     END
  END;  { quicksort }
```

H 10 L	51	45	P 29	4	100	79	T 18 R	
10 L	51	45	29	4	100	79	18	R
10	51 L	45	29	4	100	79	18	R
10	51 L	45	29	4	100	79	18 R	
10	18 L	45	29	4	100	79	51 R	
10	18	45 L	29	4	100	79	51 R	
10	18	45 L	29	4	100	79 R	51	
10	18	45 L	29	4	100 R	79	51	
10	18	45 L	29	4 R	100	79	51	
10	18	4 L	29	45 R	100	79	51	
10	18	4	29 L	45 R	100	79	51	
10	18	4	29 L R	45	100	79	51	
H 10 1	P 18	T 4	29 R	45	100	79	51	
10	18 L	4	29 R	45	100	79	51	
10	18 L	4 R	29	45	100	79	51	
10	4 L	18 R	29	45	100	79	51	

10	4	18	29	45	100	79	51	
		L						
		R						

10	4	18	29	45	100	79	51	
	R	L						

P								
H	T							
10	4	18	29	45	100	79	51	
L		R						

10	4	18	29	45	100	79	51	
L	R							

4	10	18	29	45	100	79	51	
L	R							

4	10	18	29	45	100	79	51	
	L							
	R							

4	10	18	29	45	100	79	51	
R	L							

בשלב זה מתבצע QUICKSORT על תת-תחומים: HEAD = 1 ; TAIL = 1
HEAD = 2 ; TAIL = 2
HEAD = 3 ; TAIL = 3

הפרוצדורה QUICKSORT מבצעת מיון רק אם HEAD < TAIL, ולכן לא יתבצע דבר אם ישלחו אליה הערכים שלעיל.

				H	P		T	
4	10	18	29	45	100	79	51	
				L				R

4	10	18	29	45	100	79	51	
					L			R

4	10	18	29	45	100	79	51	
					L		R	

							P	
4	10	18	29	45	51	79	100	
					L		R	

4	10	18	29	45	51	79	100	
						L	R	

4	10	18	29	45	51	79	100
							L
							R

4	10	18	29	45	51	79	100
						R	L

				H	P	T	
4	10	18	29	45	51	79	100
				L			R

4	10	18	29	45	51	79	100
					L		R

4	10	18	29	45	51	79	100
					L	R	

4	10	18	29	45	51	79	100
					L		
					R		

בשלב זה מתבצע QUICKSORT על תת-תחומים: HEAD = 5 ; TAIL = 5
HEAD = 7 ; TAIL = 7
HEAD = 8 ; TAIL = 8

הפרוצדורה QUICKSORT תבוצע רק אם HEAD < TAIL, ולכן לא יתבצע דבר אם ישלחו אליה הערכים שלעיל.

HeapSort

```
CONST
   MAX = 512;
TYPE
   ELEMENT_TYPE = INTEGER;
   INDEX = 0..MAX;
   SORT_ARRAY = ARRAY [INDEX] OF ELEMENT_TYPE;

PROCEDURE HEAP_SORT (VAR A : SORT_ARRAY;
                         N : INDEX);
   VAR
      X : ELEMENT_TYPE;
      K : INDEX;

   PROCEDURE INSERT (X : ELEMENT_TYPE;
                     I,
                     J : INDEX);
      VAR
         FATHER_INDEX,
         SON_INDEX      : INDEX;
      BEGIN  { insert }
         FATHER_INDEX := I;                      { האינדקס במערך ה"אב" }
         SON_INDEX := 2 * FATHER_INDEX; { האינדקס במערך של הבן השמאלי }
         WHILE  SON_INDEX <= J  DO
         BEGIN                                { כל עוד לא חרגנו מהמערך }
            IF  SON_INDEX < J  THEN      { אם קטן, סימן שיש גם בן ימני }
               IF  A [SON_INDEX + 1] < A [SON_INDEX]  THEN
                  SON_INDEX := SON_INDEX + 1;      { SON_INDEX יצביע }
                                                   { על הבן הקטן יותר }
            IF  X <= A [SON_INDEX]  THEN
               SON_INDEX := J + 1          { האיבר להכנסה (X) קטן מהבן }
                                           { הקטן, ויש יציאה מהלולאה. }
            ELSE
            BEGIN
               A [FATHER_INDEX] := A [SON_INDEX];          { הכנסת הבן }
                                                   { הקטן במקום האב. }
               FATHER_INDEX := SON_INDEX;        { האב הופך להיות שורש }
                      { העץ של הבן הקטן במיקום של הבן השמאלי החדש. }
               SON_INDEX := 2 * FATHER_INDEX
            END
         END;
         A [FATHER_INDEX] := X     { בשלב זה FATHER_INDEX הוא עלה או }
                                   { שורש העץ ש-X הוא הקטן ביותר מבין }
                                          { איבריו ולכן X מוכנס למקום זה. }

      END;
```

```
PROCEDURE BUILD_HEAP;
   VAR
      K : INDEX;
   BEGIN  { build }
      FOR  K := N DIV 2  DOWNTO  1  DO          { מעבר על כל ה"אבות" }
        INSERT (A [K], K, N) { A [K] הכנסה במקום המתאים בתת-עץ של }
   END;    { build }

BEGIN  { heap_sort }
   BUILD_HEAP;                                              { מעבר ראשון }
   FOR  K := N  DOWNTO  2  DO
   BEGIN
      X := A [K];
      A [K] := A [1];                       { האיבר הקטן מועבר לסוף }
      INSERT (X, 1, K - 1)        { הכנסת האיבר שהיה בסוף העץ למקום }
                                  {           המתאים, בין 1 ל-(K-1). }
   END
END;
```

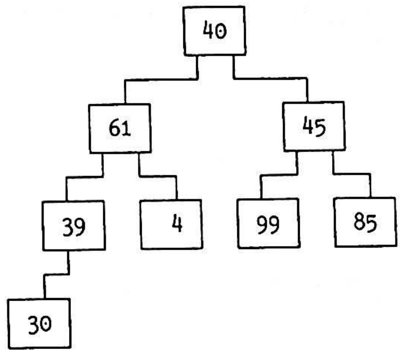
<u>הסבר:</u>

מיון זה מתייחס למערך כייצוג של עץ בינרי כמעט שלם כאשר לכל איבר במקום i הבנים שלו נמצאים במקום 2i ו-2i+1 במערך.
המספר, N DIV 2, הוא מספר האיברים שיש להם בנים (אם מספר האיברים זוגי אז לאיבר במקום n div 2 במערך יש רק בן אחד).
שלב ראשון של המיון הוא מעבר על האיברים שיש להם בנים (4 בדוגמא), מתבצע ע"י הפרוצדורה BUILD_HEAP, והכנסת כל איבר במקום המתאים בתת-העץ שהוא השורש שלו.
המעבר נעשה מהרמה העמוקה של העץ לכיוון השורש, כך שלאחר המעבר האיברים הגדולים "נדחפים" לכיוון העלים והאיברים הקטנים "צפים" לכיוון השורש, כך שכל איבר הוא הקטן ביותר בתת-העץ שהוא השורש שלו. (האיבר במקום הראשון במערך הוא הקטן ביותר בעץ).
השלב הבא הוא מעבר מסוף המערך (מקום n) עד איבר מספר 2, כאשר בכל שלב האיבר במקום n-i+1 (i=n..2) מועבר למקום ה-i והאיבר שהיה במקום ה-i מוכנס למקום המתאים בעץ בין 1 ל-(i-1).
האיבר הקטן ביותר בעץ (עד i-1) "צף" למקום הראשון במערך. התוצאה היא מערך ממויין מהגדול לקטן.

נתון המערך הבא:

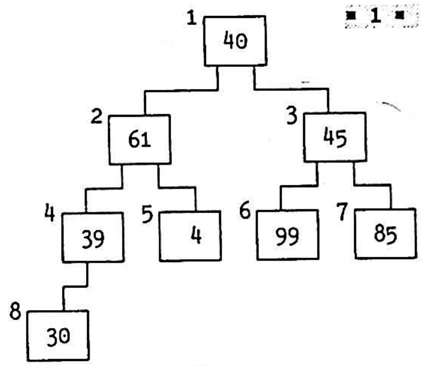
40	61	45	39	4	99	85	30

נסתכל על המערך כייצוג של עץ בינרי כמעט שלם:



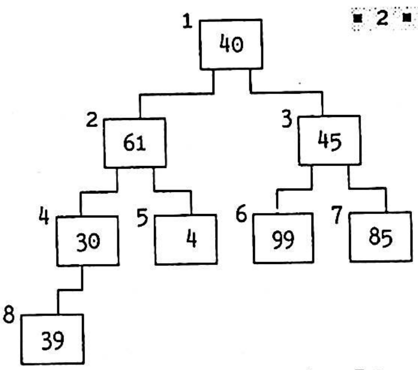
BUILD_HEAP

• 1 •



מספר האיברים N = 8

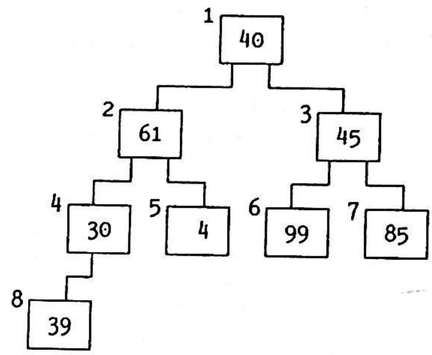
• 2 •



insert (A [k], k, n) [K = 4]

39 גדול מ-30 ולכן מתבצעת החלפה

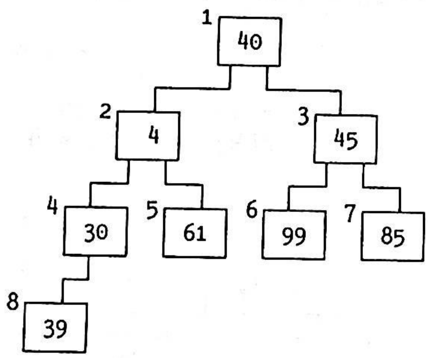
▪ 3 ▪



insert (A [k], k, n)   [k=3]

תת-העץ ששורשו A[3] לא משתנה כי
45 קטן משני בניו.

▪ 4 ▪

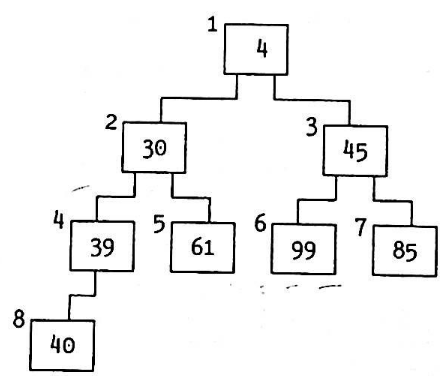


insert (A[k], k, n)   [k=2]

A[5]=4 < A[4]=30 ולכן A[5] מוחלף
ב-A[2].

▪ 5 ▪

במעבר הראשון ב-insert
הערך 4 שב-A[2] מוכנס ל-A[1]
במעבר השני הערך 30 שב-A[4] מוכנס ל-A[2]
במעבר השלישי הערך 39 שב-A[8] מוכנס ל-A[4]
לאחר יציאה מוכנס הערך 40 שב-A[1] ל-A[8]
ומתקבל:
שים לב! בשלב זה כל איבר הוא הקטן ביותר
בתת-העץ שלו.

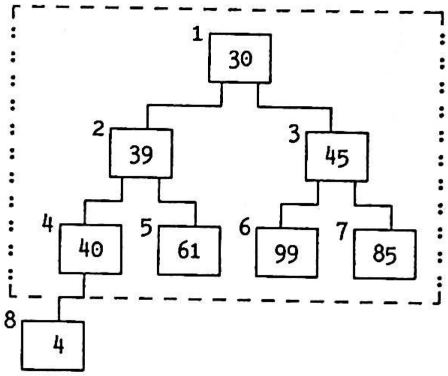


insert (A[k], k, n)   [k=1]

התחום במערך (עץ) ש-insert פועל עליו מסומן בקו מקווקו.

■ 1 ■

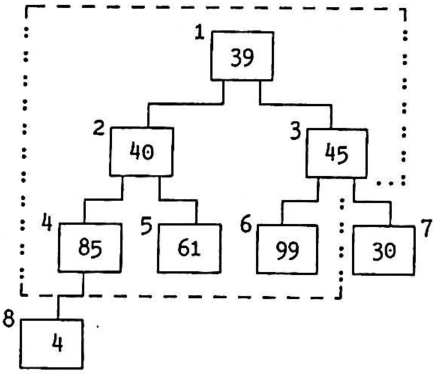
k = n = 8
x := A[k] = 40
A[k] := A[1] = 4



insert (x, 1, 7)
האיבר X מוכנס לעץ במקומות 1 עד 7
(מסומן במסגרת מקווקות).

■ 2 ■

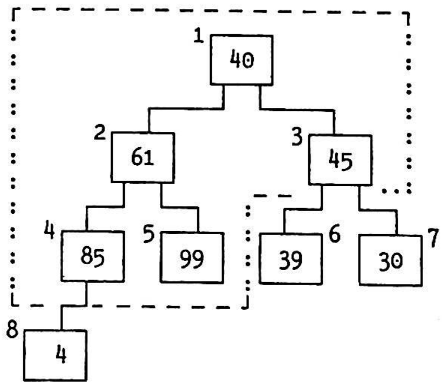
k = 7
x := A[k] = 85
A[k] := A[1] = 30



insert (x, 1, 6)

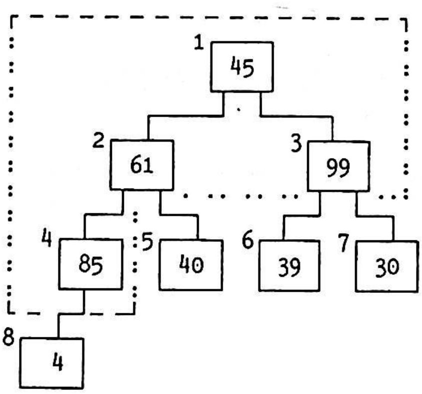
■ 3 ■

k = 6
x := A[k] = 99
A[k] := A[1] = 39
insert (x, 1, 5)



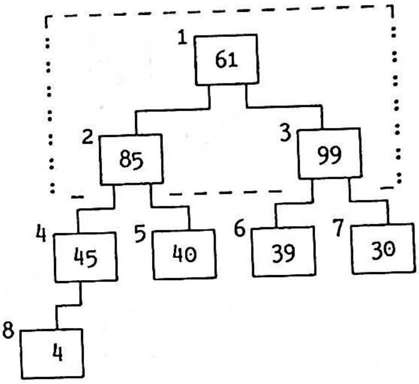
■ 4 ■

k = 5
x := A[k] = 99
A[k] := A[1] = 40
insert (x, 1, 4)



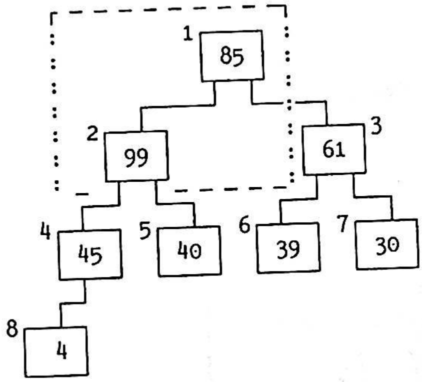
■ 5 ■

```
k = 4
x := A[k] = 85
A[k] := A[1] = 45
insert (x, 1, 3)
```



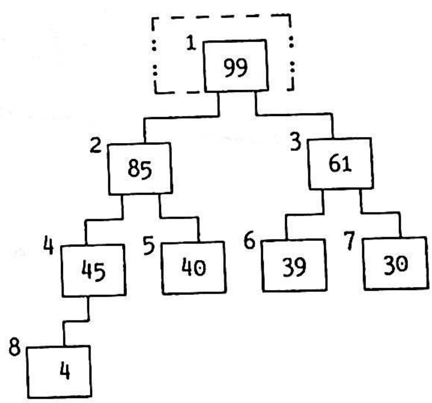
■ 6 ■

```
k = 3
x := A[k] = 99
A[k] := A[1] = 61
insert (x, 1, 2)
```



■ 7 ■

```
k = 2
x := A[k] = 99
A[k] := A[1] = 85
insert (x, 1, 1)
```



מערך:

99	85	61	45	40	39	30	4
1							8



"חיפוש"

פרק 8
פרוצדורות ופונקציות

שאלות

שאלה 1

א. פלט התוכנית:

```
19 17
15 20
```

ב. פלט התוכנית:

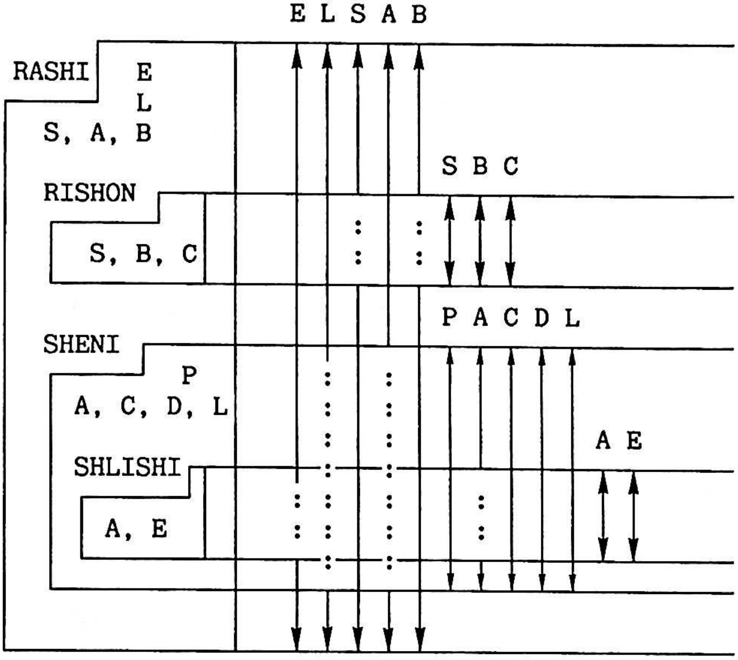
```
19 17
19 20
```

ג. פלט התוכנית:

```
19 17
19 17
```

שאלה 2

את התשובה לשאלה זו נציג בתרשים (ראה גם סעיף 8.4 בספר פסקל).





שאלה 3

בתרגיל זה ישנם מספר משתנים, שאינם מקבלים ערכים כלשהם בתוכנית, אך הם מודפסים. דבר זה יגרום להופעת מספרים או תווים אקראיים במהלך הביצוע.

להלן טבלה שמרכזת את הפלט:

A	B	C	D	X	Y
10	ל"מ	ל"מ	'A'	10	20
20	CHR(20)	CHR(20)	'A'	ל"מ	ל"מ
20	-"-	-"-	'A'	20	0
20	ל"מ	ל"מ	'A'	20	30
20	CHR(20)	CHR(30)	'A'	ל"מ	ל"מ
20	-"-	-"-	'A'	30	0
20	ל"מ	ל"מ	'A'	30	20
20	CHR(20)	CHR(20)	'A'	ל"מ	ל"מ
20	-"-	-"-	'A'	20	0

מקרא: ל"מ - פרושו "לא מוגדר"

תוכניות

תוכנית 1

```
PROGRAM TAX_CALCULATING (INPUT, OUTPUT);
VAR
   STATUS       : CHAR;
   SALARY, TAX : REAL;

FUNCTION PAIR_TAX (S : REAL) : REAL;
    BEGIN
      IF  S > 1000  THEN
         PAIR_TAX := 0.2 * 1000 + (S - 1000) * 0.4
      ELSE
         PAIR_TAX := S * 0.2
    END;    { pair_tax }

FUNCTION SINGLE_TAX (S : REAL) : REAL;
    BEGIN
      IF  S > 500  THEN
         SINGLE_TAX := 0.1 * 500 + (S - 500) * 0.6
      ELSE
         SINGLE_TAX := S * 0.1
    END;    { single_tax }

BEGIN  { main }
   WRITELN ('מצב משפחתי? N = נשוי ,  B = בודד');
   READLN (STATUS);
   WRITELN ('רמת הכנסה');
   READLN (SALARY);
   IF  STATUS = 'N'  THEN
      TAX := PAIR_TAX (SALARY)
   ELSE
      TAX := SINGLE_TAX (SALARY);
   WRITELN (TAX:4:2, 'שיעור המס הוא ')
END.
```

הערה:

* הפונקציה PAIR_TAX מחשבת מס עבור זוג נשוי.
* הפונקציה SINGLE_TAX מחשבת מס עבור אדם בודד.

תוכנית 2

```
PROGRAM HISTOGRAM (INPUT, OUTPUT);
VAR
   X, DIGITS : INTEGER;

PROCEDURE DRAW_LINE (N : INTEGER);
   VAR
      I :INTEGER;
    BEGIN
      FOR  I := 1  TO  N  DO
         WRITE ('-');
      WRITELN
    END;     { draw_line }

BEGIN    { main }
   WRITE ('X = ');
   READLN (X);
   WRITE (X, '  ');
   DIGITS := 0;
   WHILE  ABS (X) >= 1 DO
    BEGIN
      DIGITS := DIGITS + 1;
      X := X DIV 10
    END;
   DRAW_LINE (DIGITS)
END.
```

תוכנית 4

```
PROGRAM SQUARE_MATRIX (INPUT, OUTPUT);
CONST
   MAX_ROW    = 3;
   MAX_COLUMN = 4;
TYPE
   MATRIX = ARRAY [1..MAX_ROW, 1..MAX_COLUMN] OF REAL;
VAR
   A                 : MATRIX;
   INDEX_1, INDEX_2  : INTEGER;

FUNCTION SCHUM (SUG : CHAR;
                VAR NETUNIM : MATRIX;
                N : INTEGER ) : REAL;
   VAR
      I   : INTEGER;
      SUM : REAL;
    BEGIN
      SUM := 0;
```

```
(18)        IF  SUG = 'A'  THEN              { חישוב סכום הריבועים בעמודה }
(19)           FOR  I := 1  TO  MAX_ROW  DO
(20)              SUM := SUM + SQR (NETUNIM [I, N])
(21)        ELSE                             { חישוב סכום הריבועים בשורה }
(22)           FOR  I := 1  TO  MAX_COLUMN  DO
(23)              SUM := SUM + SQR (NETUNIM [N, I]);
(24)        SCHUM := SUM
(25)      END;

(26) BEGIN  { main }
(27)     FOR  INDEX_1 := 1  TO  MAX_ROW  DO
(28)      BEGIN
(29)        WRITELN (INDEX_1:1, 'הכנס שורה מספר ');
(30)        FOR  INDEX_2 := 1  TO  MAX_COLUMN  DO
(31)           READ (A [INDEX_1, INDEX_2]);
(32)        WRITELN
(33)      END;
(34)     FOR  INDEX_1 := 1  TO  MAX_ROW  DO
(35)      BEGIN
(36)        FOR  INDEX_2 := 1  TO  MAX_COLUMN  DO
(37)           WRITE (A [INDEX_1, INDEX_2]:10:2);
(38)        WRITELN (SCHUM ('S', A, INDEX_1):10:2)
(39)      END;
(40)     FOR  INDEX_2 := 1  TO  MAX_COLUMN  DO
(41)        WRITE (SCHUM ('A', A, INDEX_2):10:2)
(42) END.
```

הסבר:

27-33 קריאת המטריצה.
34-39 הדפסת שורות המטריצה ולאחריהן - תוצאת החישוב הנדרש.
40-41 הדפסת ערכי החישוב לעמודות המטריצה.

תוכנית 6

```
PROGRAM STRINGS (INPUT, OUTPUT);
CONST
   MAX_CHR = 80;
TYPE
   STR = PACKED ARRAY [1..MAX_CHR] OF CHAR;
VAR
   STR_A, STR_B : STR;
   M1, M2  : INTEGER;
```

```
PROCEDURE SUB_STR (N1, N2 : INTEGER
                   VAR STR1, STR2 : STR);
   VAR
      I : INTEGER;
    BEGIN
      FOR  I := N1  TO  N1 + N2 - 1  DO { העתקת המחרוזת המבוקשת }
         STR2 [I - N1 + 1] := STR1 [I];
      FOR  I := N2 + 1  TO  MAX_CHR  DO {     מילוי שאר המקומות }
         STR2 [I] := ' '                {        ב-STR2 ברווחים }
    END;

BEGIN  { main }
   READLN (STR_A);
   READLN (M1, M2);
   SUB_STR (M1, M2, STR_A, STR_B);
   WRITELN (STR_B)
END.
```

הסבר:

הפרוצדורה SUB_STR מחזירה ב-STR2 את התווים של SRT1 הנמצאים ממקום N1 באורך N2.
גודל המחרוזת המכסימלי, MAX_CHR, נקבע לפי היישום ולכן לא נבדקת אפשרות של חריגה מהמערך.

תוכנית 7

```
FUNCTION POWER (X, Y : REAL) : REAL;
BEGIN
   POWER := EXP (Y * LN (X))
END;
```

הסבר:

לפי $X^Y=e^{(Y*ln(X))}$. ראה הסבר בעמוד 28-29 בספר פסקל.

תוכנית 8

```
PROGRAM MARK_SORT (INPUT, OUTPUT);
CONST
   MAX_STUDENT = 200;
TYPE
   NAME  = PACKED ARRAY [1..20] OF CHAR;
   MARKS = ARRAY [1..MAX_STUDENT] OF INTEGER;
VAR
```

```
   NAMES : ARRAY [1..MAX_STUDENT] OF NAME;
   MARK  : MARKS;
   N, I  : INTEGER;
   A     : REAL;
                                        { הפונקציה AVERAGE מחזירה את הממוצע של }
FUNCTION AVERAGE : REAL;                { מערך MARK, שהוא משתנה גלובלי בתוכנית. }
   VAR
      I   : INTEGER;
      SUM : REAL;
   BEGIN
      SUM := 0;
      FOR  I := 1  TO  N  DO
         SUM := SUM + MARK [I];
      AVERAGE := SUM / N
   END;
                                  { הפונקציה STANDARD_DEVIATION מחזירה את סטיית }
FUNCTION STANDARD_DEVIATION : REAL;       { התקן של הערכים במערך MARK. }
   VAR
      I       : INTEGER;
      SQR_SUM : REAL;
   BEGIN
      SQR_SUM := 0;
      FOR  I := 1  TO  N  DO
         SQR_SUM := SQR_SUM + SQR (MARK [I]);
      STANDARD_DEVIATION := SQRT (SQR_SUM / N - SQR (AVERAGE));
   END;

PROCEDURE SORT;        { הפונקציה SORT מבצעת מיון בועות של מערך הציונים }
   VAR
      L, I, J   : INTEGER;
      TEMP_MARK : INTEGER;
      TEMP_NAME : NAME;
   BEGIN
      J := N;
      WHILE  J > 0  DO
       BEGIN
         L := 0;
         FOR  I := 1  TO  J - 1  DO
            IF  MARK [I] < MARK [I + 1]  THEN
             BEGIN
               TEMP_MARK     := MARK [I];
               MARK [I]      := MARK [I + 1];
               MARK [I + 1]  := TEMP_MARK;
               TEMP_NAME     := NAMES [I];
               NAMES [I]     := NAMES [I + 1];
               NAMES [I + 1] := TEMP_NAME;
               L := I
             END;
         J := L
      END    { while j > 0 }
   END;
```

```
BEGIN { main }
   WRITELN ('מספר הסטודנטים');
   READLN (N);
   WRITELN ('שם הסטודנט        ציון ');
   FOR  I := 1  TO  N  DO
      READLN (MARK [I], NAMES [I]);
   SORT;
   A := AVERAGE;
   WRITELN ('סטיה מהממוצע         ציון              שם תלמיד':55);
   FOR  I := 1  TO  N  DO
      WRITELN (NAMES [I]:25, MARK [I]:10, MARK [I] - A:15:2);
   WRITELN;
   WRITELN (STANDARD_DEVIATION:4:2, ' סטית התקן ', A:12:2, ' ממוצע')
END.
```

תוכנית 9 (א' ו-ב')

```
(1)  PROGRAM TRIANGLE (INPUT, OUTPUT);
(2)  VAR
(3)     I, TOP, HEIGHT : INTEGER;

(4)  PROCEDURE OTAKIM (C : CHAR;
(5)                    N : INTEGER);
(6)     VAR
(7)        I : INTEGER;
(8)      BEGIN
(9)        FOR  I := 1  TO  N  DO
(10)          WRITE (C)
(11)     END;

(11) BEGIN { main }
(12)    READLN (HEIGHT, TOP);
(13)    FOR  I := 0  TO  HEIGHT - 1  DO
(14)     BEGIN
(15)       OTAKIM (' ', TOP - I);
(16)       OTAKIM ('*', 1 + 2 * I);
(17)       WRITELN
(18)     END
(19) END.
```

<u>הסבר:</u>

הפרוצדורה OTAKIM (4-11) מקבלת תו (C) ומספר שלם (N), ומדפיסה N פעמים את התו C.
בכל שורה של המשולש מודפסים רווחים שמספרם יורד ב-1 בכל שורה (16).
לאחר מכן מודפסים '*' במספר העולה ב-2 בכל שורה (17).
גובה המשולש (HEIGHT) ומספר הרווחים לפני הקודקוד העליון (TOP) הינם נתוני קלט.

תוכנית 9 (ג' ו-ד')

```
(1)  PROGRAM HEADER (INPUT, OUTPUT);
(2)  CONST
(3)     MAX_CHR = 60;
(4)  TYPE
(5)     MARACH = PACKED ARRAY [1..MAX_CHR] OF CHAR;
(6)  VAR
(7)     MARACH_60 : MARACH;
(8)     I : INTEGER;

(9)  FUNCTION TAV_ACHRON (VAR  MARACH_60 : MARACH) : INTEGER;
(10)    VAR
(11)       I : INTEGER;
(12)    BEGIN
(13)       I := MAX_CHR;
(14)       WHILE  MARACH_60 [I] = ' '  DO
(15)          I := I - 1;
(16)       TAV_ACHRON := I
(17)    END;

(18) PROCEDURE OTAKIM (C : CHAR;   N : INTEGER);
(19)    VAR
(20)       I : INTEGER;
(21)    BEGIN
(22)       FOR  I := 1  TO  N  DO
(23)          WRITE (C)
(24)    END;

(25) BEGIN { main }
(26)    READLN (MARACH_60);
(27)    OTAKIM (' ', 30 - TAV_ACHRON (MARACH_60) DIV 2);
(28)    OTAKIM ('*', 10);
(29)    FOR  I := 1  TO  TAV_ACHRON (MARACH_60)  DO
(30)       WRITE (MARACH_60 [I]);
(31)    OTAKIM ('*', 10)
(32) END.
```

<u>הסבר:</u>

הפונקציה TAV_ACHRON מוצאת את התו האחרון במערך ע"י סריקה נסוגה מסוף המערך (כלומר מן הסוף להתחלה).
שים לב, שהערך המכסימלי שתחזיר הפונקציה TAV_ACHRON יהיה 60, כדי שיהיה ניתן להדפיס בשורה אחת מחרוזת זו ועוד 20 כוכביות (10 משמאל ו-10 מימין). הפונקציה מחזירה למעשה את אורך המחרוזת.

27 הדפסת רווחים לפני השורה. הדרך - מציאת התווים בכותרת וחלוקה לשניים על ידי DIV והחסרה מ-30, שזהו מיקום מרכז הכותרת.
28 הדפסת '*' 10 פעמים לפני הכותרת.
29-30 הדפסת השורה.
31 הדפסה של התו '*' פעמים לאחר הכותרת.

תוכנית 10

תשובה לסעיף א': שורות 13-24
תשובה לסעיף ב': שורות 25-44
תשובה לסעיף ג': התוכנית כולה.

```
(1)  PROGRAM  NAME_NUMBER (INPUT, OUTPUT);
(2)  CONST
(3)     NAME_COLUMN   = 10;
(4)     NUMBER_COLUMN = 28;
(5)     MAX_LENGTH    = 80;
(6)     ERROR         = -1;
(7)  TYPE
(8)     LINE = PACKED ARRAY [1..MAX_LENGTH] OF CHAR;
(9)  VAR
(10)    I, J, K : INTEGER;
(11)    PAGE : ARRAY [1..50] OF LINE;
(12)    NUMBERS, NAME_LENGTH : ARRAY [1..50] OF INTEGER;

(13) FUNCTION ORECH (X :INTEGER) : INTEGER;
(14)    VAR
(15)       DIGITS :INTEGER;
(16)    BEGIN
(17)       DIGITS := 0;
(18)       WHILE  X >= 1 DO
(19)        BEGIN
(20)          DIGITS := DIGITS + 1;
(21)          X := X DIV 10
(22)        END;
(23)       ORECH := DIGITS
(24)    END;

(25) FUNCTION KRA_AD_PSIK (VAR  NAME : LINE) : INTEGER;
(26)    VAR
(27)       I  : INTEGER;
(28)       CH : CHAR;
(29)    BEGIN
(30)       I := 1;
(31)       READ (CH);
(32)       WHILE  (CH <> ',') AND NOT EOLN  DO
(33)        BEGIN
(34)          NAME [I] := CH;
(35)          READ (CH);
(36)          I := I + 1
(37)        END;
(38)       IF  EOLN  THEN
(39)        BEGIN
(40)          WRITELN ('שגיאה! - חסר פסיק');
(41)          I := 0
(42)        END;
(43)       KRA_AD_PSIK := I - 1   { אם היתה שגיאה - I קיבל את הערך 0, }
```

```
(44)    END;                    { ולכן הפונ' תחזיר ערך -1, כלומר ERROR }

(45) PROCEDURE OTAKIM (C : CHAR;
(46)                   N : INTEGER);
(47)    VAR
(48)       I : INTEGER;
(49)    BEGIN
(50)       FOR  I := 1  TO  N  DO
(51)          WRITE (C)
(52)    END;

(53) BEGIN   { main }
(54)    J := 0;
(55)    REPEAT
(56)       J := J + 1;
(57)       NAME_LENGTH [J] := KRA_AD_PSIK (PAGE [J]);
(58)       IF  NAME_LENGTH [J] <> ERROR  THEN
(59)          READLN (NUMBERS [J])
(60)       ELSE
(61)          READLN
(62)    UNTIL  (PAGE [J, 1] = '.') OR (NAME_LENGTH [J] = ERROR);
(63)    FOR  I := 1  TO  J - 1  DO
(64)     BEGIN
(65)       OTAKIM (' ', 10);
(66)       FOR  K := 1  TO  NAME_LENGTH [I]  DO
(67)          WRITE (PAGE [I, K]);
(68)          OTAKIM (' ', NUMBER_COLUMN - NAME_COLUMN -
(69)                         NAME_LENGTH [I] - ORECH (NUMBERS [I]));
(70)          WRITELN (NUMBERS [I]:ORECH (NUMBERS [I]))
(71)     END
(72) END.
```

<u>הסבר:</u>

הפונקציה ORECH (13-24) מקבלת מספר שלם ומחזירה את מספר הספרות במספר. הפונקציה KRA_AD_PSIK מחזירה את מספר התווים שנקראו עד פסיק. היא מחזירה את הערך ERROR (המוגדר -1 בשורה 6). אם אין פסיק לפני סוף שורה, התווים שנקראו מוחזרים במשתנה NAME.

54-62 קריאת הנתונים עד שם שמתחיל בנקודה, או עד שגיאה בנתונים. אורך כל שם מחושב על ידי הפונקציה KRA_AD_PSIK ומוחסן במערך NAME_LENGHT במקום המתאים.

64 הדפסת של 10 רווחים.

65-66 הדפסת השם.

67-68 הדפסת רווחים עד להדפסת המספר.

69 הדפסת המספר בשדה לפי מספר הספרות המרכיבות אותו.

```
PROGRAM STRINGS (INPUT, OUTPUT);
TYPE
   MACHROZET = PACKED ARRAY [1..100] OF CHAR;
VAR
   STR, NEW_STR, PATTERN  : MACHROZET;
   LENGTH, NEW_STR_LENGTH, PATTERN_LENGTH, I : INTEGER;

PROCEDURE KRA_AD_GVUL (VAR M      : MACHROZET;
                       VAR ORECH : INTEGER);
   VAR
      CH : CHAR;
   BEGIN
      ORECH := 0;
      READ (CH);
      WHILE  CH <> '/'  DO
       BEGIN
         ORECH := ORECH + 1;
         M [ORECH] := CH;
         READ (CH)
       END;
      WRITELN
   END;

FUNCTION TEUM (M1,
               M2 : MACHROZET;
               N1,
               N2 : INTEGER   ) : BOOLEAN;
   VAR
      I     : INTEGER;
      FOUND : BOOLEAN;
   BEGIN
      FOUND := TRUE;                { ערך התחלתי - המחרוזות תואמות }
      I := 0;
      WHILE (I < N1) AND FOUND  DO
       BEGIN
         IF  M1 [N2 + I] <> M2 [1 + I]  THEN
            FOUND := FALSE;         { המחרוזות אינן תואמות }
         I := I + 1
       END;
      TEUM := FOUND
   END;

PROCEDURE MECHIKA (VAR M       : MACHROZET;
                   VAR ORECH   : INTEGER;
                       N1, N2  : INTEGER);
   VAR
      I : INTEGER;
   BEGIN
      FOR  I := N1  TO  ORECH - N2  DO
```

```
(47)          M [I] := M [I+N2];
(48)       ORECH := ORECH - N2          { עדכון אורך המחרוזת לאחר מחיקה }
(49)    END;

(50) PROCEDURE HOSAFA (VAR M1,
(51)                       M2      : MACHROZET;
(52)                   VAR ORECH   : INTEGER;
(53)                       N1, N2  : INTEGER);
(54)    VAR
(55)       I : INTEGER;
(56)    BEGIN
(57)       FOR  I := ORECH + N2  DOWNTO  N1 + N2  DO
(58)          M1 [I] := M1 [I - N2];  { הכנת מקום ריק למחרוזת שתתוסף }
(59)       ORECH := ORECH + N2;                 { עדכון אורך המחרוזת }
(60)       FOR  I := 1  TO  N2  DO
(61)          M1 [N1 - 1 + I] := M2 [I]         { שתילת המחרוזת למקום }
(62)    END;

(63) BEGIN { main }
(64)    WRITELN ('הכנס מחרוזת');
(65)    KRA_AD_GVUL (STR, LENGTH);
(66)    WRITELN ('הכנס מחרוזת לזיהוי');
(67)    KRA_AD_GVUL (PATTERN, PATTERN_LENGTH);
(68)    WRITELN ('הכנס מחרוזת להחלפה');
(69)    KRA_AD_GVUL (NEW_STR, NEW_STR_LENGTH);
(70)    I := 1;
(71)    WHILE  I <= LENGTH  DO
(72)     BEGIN
(73)       IF  TEUM (STR, PATTERN, PATTERN_LENGTH, I)  THEN
(74)        BEGIN
(75)          MECHIKA (STR, LENGTH, I, PATTERN_LENGTH);
(76)          HOSAFA (STR, NEW_STR, LENGTH, I, NEW_STR_LENGTH);
(77)          I := I + NEW_STR_LENGTH - PATTERN_LENGTH
(78)        END;
(79)       I := I + 1
(80)     END;
(81)    WRITELN ('המחרוזת לאחר החלפה');
(82)    FOR  I := 1  TO  LENGTH  DO
(83)       WRITE (STR [I])
(84) END.
```

הסבר:

71-80 חיפוש לאורך המחרוזת עד שמוצאים מקום, שבו נמצא ה-PATTERN. אם הוא נמצא (67-71) - אז מוחקים אותו מהמחרוזת (68) ומוסיפים את המחרוזת החדשה במקומו (69). מעדכנים את המצביע למיקום במחרוזת (I), כדי שיפנה לתו האחרון של המחרוזת שהוספנו (70). לאחר ביצוע שורה 72, I מצביע על התו הבא.

תוכנית 12

```
PROGRAM CONVERT (INPUT, OUTPUT);
CONST
   END_OF_INPUT = 0;
VAR
   SUM, X : INTEGER;

FUNCTION OCTAL_TO_DECIMAL (N :INTEGER) : INTEGER;
   VAR
      SUM, POWER : INTEGER;
   BEGIN
      POWER := 1;
      SUM   := 0;
      WHILE  N > 0  DO
       BEGIN
         SUM := SUM + POWER * (N MOD 10);
         POWER := POWER * 8;
         N := N DIV 10
       END;
      OCTAL_TO_DECIMAL := SUM
   END;

FUNCTION DECIMAL_TO_OCTAL (N :INTEGER) : INTEGER;
   VAR
      SUM, POWER : INTEGER;
   BEGIN
      POWER := 1;
      SUM   := 0;
      WHILE  N > 0  DO
       BEGIN
         SUM := SUM + POWER * (N MOD 8);
         POWER := POWER * 10;
         N := N DIV 8
       END;
      DECIMAL_TO_OCTAL := SUM
   END;

BEGIN { main }
   SUM := 0;
   REPEAT
      READLN (X);
      SUM := SUM + OCTAL_TO_DECIMAL (X)
   UNTIL  X = END_OF_INPUT;
   WRITELN (DECIMAL_TO_OCTAL (SUM))
END.
```

<u>הסבר:</u>

הפונקציה OCTAL_TO_DECIMAL מקבלת מספר אוקטלי ומחזירה מספר עשרוני. בכל שלב מבודדים את הספרה הימנית ביותר (N MOD 10), כופלים אותה בחזקה המתאימה של 8 (POWER יקבל את הערכים: 1, 8, 64 ...) ומוסיפים לסכום המצטבר. לאחר מכן מחלקים את N (חלוקה בשלמים) ב-10.

הפונקציה DECIMAL_TO_OCTAL פועלת באופן דומה, אך הפעם החלוקה בכל שלב היא ב-8 וההכפלה היא בחזקות של 10 (POWER יקבל ערכים: 1, 10, 100 ...). החיבור מתבצע בשורה 38 עד שמגיע נתון 0 המסמן את סוף הקלט.

<u>הערה:</u> מספר אוקטלי מכיל את הספרות 0 עד 7 בלבד. ההנחה בתוכנית היא, שהמספר האוקטלי הינו חוקי.

תוכנית 13

```
(1)  PROGRAM CONVERT (INPUT, OUTPUT);
(2)  CONST
(3)     END_OF_INPUT = 0;
(4)  VAR
(5)     SUM, X : INTEGER;

(6)  FUNCTION OCTAL_TO_DECIMAL (N :INTEGER) : INTEGER;
(7)     VAR
(8)        SUM, POWER : INTEGER;
(9)     BEGIN
(10)       POWER := 1;
(11)       SUM := 0;
(12)       WHILE  N > 0  DO
(13)        BEGIN
(14)          SUM := SUM + POWER * (N MOD 10);
(15)          POWER := POWER * 8;
(16)          N := N DIV 10
(17)        END;
(18)       OCTAL_TO_DECIMAL := SUM
(19)    END;

(20) FUNCTION ADD_OCTAL (N1, N2 : INTEGER) : INTEGER;
(21)    VAR
(22)       SUM, POWER, CARRY : INTEGER;
(23)    BEGIN
(24)       POWER := 1;
(25)       SUM   := 0;
(26)       CARRY := 0;
(27)       WHILE  (N1 > 0) OR (N2 > 0) OR (CARRY > 0) DO
(28)        BEGIN
(29)          SUM   := SUM + POWER *
(30)                   (CARRY + ((N1 MOD 10 + N2 MOD 10) MOD 8));
```

```
(31)          CARRY := (CARRY + N1 MOD 10 + N2 MOD 10) DIV 8;
(32)          POWER := POWER * 10;
(33)          N1 := N1 DIV 10;
(34)          N2 := N2 DIV 10
(35)        END;    { while }
(36)       ADD_OCTAL := SUM
(37)    END;

(38) BEGIN { main }
(39)    SUM := 0;
(40)    REPEAT
(41)       READLN (X);
(42)       SUM := ADD_OCTAL (X, SUM);
(43)    UNTIL  X = END_OF_INPUT;
(44)    WRITELN (SUM, ' : סכום אוקטלי ',
(45)       OCTAL_TO_DECIMAL (SUM):20, ' : סכום עשרוני')
(46) END.
```

<u>הסבר:</u>

* הפונקציה ADD_OCTAL מחזירה סכום של שני מספרים אוקטליים.
* המשתנה CARRY מקבל בכל שלב של חיבור את הנשא כתוצאה מחיבור שתי ספרות של המחוברים.
* החיבור מתבצע כל עוד קיים מספר שאינו 0 או כשקיים נשא (carry).
* כאשר יש נשא הוא מחובר לסכום שתי הספרות בשלב הבא.

דוגמה: 46 + 42 (שני המספרים אוקטליים).

```
(I)     42
      + 46         שורה 29        שורה 31
        0 --->> sum = 0    ;  carry = 1

(II)    42
      + 46
       1  --->> sum = 10   ; carry = 1

(III)   42
      + 46
       1  --->> sum = 110 ; carry = 0
```

תוכנית 14

```
PROGRAM CALCULATE_EXPONENT (OUTPUT);
CONST
   JUMP = 0.01;
VAR
   X : REAL;
```

```
(6)  FUNCTION  EXPONENT (X : REAL) : REAL;
(7)     CONST
(8)        EPSILON = 0.0001;
(9)     VAR
(10)       SUM, TEMP_SUM, POWER, FACT, INDEX : REAL;
(11)    BEGIN
(12)       SUM      := 1;
(13)       TEMP_SUM := 0;
(14)       INDEX    := 1;
(15)       FACT     := 1;
(16)       POWER    := 1;
(17)       REPEAT
(18)          SUM   := SUM + TEMP_SUM;    {              סכום איברי הטור }
(19)          FACT  := FACT * INDEX;      { 1!, 2!, 3!, 4!...               }
(20)          POWER := POWER * X;         { X, X^2, X^3, X^4 ...            }
(21)          INDEX := INDEX + 1;         { מונה מספר האיברים בטור + 1 }
(22)          TEMP_SUM := POWER / FACT    {             חישוב לאיבר בודד }
(23)       UNTIL  ABS (TEMP_SUM) < ABS (EPSILON * SUM);
(24)       EXPONENT := SUM
(25)    END;

(26) BEGIN  { main }
(27)    X := 0;
(28)    REPEAT
(29)       WRITELN ('X=', X:4:2, 'EXP(X)=':20, EXPONENT (X):4:2);
(30)       X := X + JUMP
(31)    UNTIL  X > 1
(32) END.
```

__הסבר:__

הפונקציה EXPONENT מקבלת מספר ממשי X ומחזירה קירוב לערך e^x.

19 המשתנה FACT הוא INDEX! (עצרת).

20 המשתנה POWER הוא X^{INDEX} (חזקה).

22 המשתנה TEMP_SUM
מקבל בכל שלב את האיבר הבא של הטור (X^{INDEX}/INDEX!).
סיכום האיברים (18) מתבצע כל עוד השינוי היחסי קטן מהערך
EPSILON=0.0001.

תוכנית 15

```
(1)  PROGRAM TESTS (INPUT, OUTPUT);
(2)  CONST
(3)     MAX_QUESTION = 50;
(4)     MAX_STUDENT  = 200;
(5)  TYPE
(6)     ANSWERS = ARRAY [1..MAX_QUESTION] OF 1..4;
(7)     NUMBERS = ARRAY [1..MAX_STUDENT]  OF REAL;
(8)     MARKS   = ARRAY [1..MAX_STUDENT]  OF INTEGER;
(9)  VAR
(10)    KEY : ANSWERS;
(11)    ANSWER_LIST : ARRAY [1..MAX_STUDENT] OF ANSWERS;
(12)    FINAL_MARK  : MARKS;
(13)    ID  : NUMBERS;
(14)    I, J, N, Q  : INTEGER;

     { הפונקציה MARK מקבלת את מספר השאלות (N),
       את מערך התשובות (ANSWER) ואת מערך התשובות הנכונות (KEY).
       היא מחזירה את הציון הממוצע על פי מספר התשובות הנכונות (24). }

(15) FUNCTION MARK (N          : INTEGER;
(16)                VAR ANSWER, KEY    : ANSWERS) : INTEGER;
(17)    VAR
(18)       I, RIGHT : INTEGER;
(19)    BEGIN
(20)       RIGHT := 0;
(21)       FOR  I := 1  TO  N  DO
(22)          IF  ANSWER [I] = KEY [I]  THEN
(23)             RIGHT := RIGHT + 1;        { ספירה של התשובות הנכונות }
(24)       MARK := ROUND (100 * RIGHT / N)       { חישוב הציון הממוצע }
(25)    END;

(26) BEGIN { main }
(27)    WRITELN ('מספר השאלות? ');
(28)    READLN (Q);
(29)    WRITELN ('מפתח התשובות? ');
(30)    FOR  I := 1  TO  Q  DO
(31)       READLN (KEY [I]);
(32)    WRITELN ('מספר הסטודנטים? ');
(33)    READLN (N);
(34)    FOR  I := 1  TO  N  DO
(35)     BEGIN
(36)       WRITELN ('מספר הסטודנט? ');
(37)       READLN (ID [I]);
(38)       WRITELN ('תשובותיו לשאלות? ');
(39)       FOR  J := 1  TO  Q  DO
(40)          READLN (ANSWER_LIST [I, J])
(41)     END;
(42)    WRITELN ('מפתח המבחן':30);
(43)    FOR  J := 1  TO  Q  DO
```

```
(44)         WRITE (KEY [J]:3);
(45)      WRITELN;
(46)      WRITELN ('מספר תלמיד   ציון              תשובות');
(47)      FOR  I := 1  TO  N  DO
(48)       BEGIN
(49)         WRITE (ID [I]:10:0, MARK (Q, ANSWER_LIST [I], KEY):7);
(50)         FOR  J := 1  TO  Q  DO
(51)            WRITE (ANSWER_LIST [I, J]:3);
(52)         WRITELN
(53)       END
(54) END.
```

<u>הסבר:</u>

27-41 קריאת הקלט.

42-44 הדפסת מפתח המבחן בראש הדו"ח.

47-53 הדפסת מספר תלמיד וציון עבור כל אחד מהתלמידים. משתמשים לשם כך בפונקציה MARK (48).

תוכנית 16

```
PROGRAM PERMUTATION (INPUT, OUTPUT);
VAR
   X, N : REAL;

FUNCTION FACT (N : REAL) : REAL;
   VAR
      SUM : REAL;
   BEGIN
      SUM := 1;
      WHILE  N > 1  DO
       BEGIN
         SUM := SUM * N;
         N := N - 1
       END;
      FACT := SUM
   END;

BEGIN  { main }
   READLN (N, X);
   WRITELN (FACT (N) / (FACT (X) * FACT (N - X)):10:0)
END.
```

<u>הסבר:</u>

הפונקציה FACT (N) מחזירה ערך של !N.
N! (N עצרת) מחושב כך: N = 2 * ... * (N-2) * (N-1) * N

<u>הערה:</u> המשתנה N מוגדר REAL כדי שלא תיווצר גלישה.

שים לב שערכו של N בשורה 18 לא משתנה אף על פי שהוא משתנה בתוך הפונקציה (12), כי N הוא פרמטר ערכי ולא פרמטר משתנה.

תוכנית 17

```
PROGRAM PRICES (INPUT, OUTPUT);
VAR
   PRICE : REAL;
   CH    : CHAR;

PROCEDURE NUMBER_WORD (N : INTEGER);
    BEGIN
      CASE  N  OF
         1 : WRITE ('ONE ');
         2 : WRITE ('TWO ');
         3 : WRITE ('THREE ');
         4 : WRITE ('FOUR ');
         5 : WRITE ('FIVE ');
         6 : WRITE ('SIX ');
         7 : WRITE ('SEVEN ');
         8 : WRITE ('EIGHT ');
         9 : WRITE ('NINE ')
      END
    END;

PROCEDURE TENS_WORD (N : INTEGER);
    BEGIN
      CASE  N  OF
         10 : WRITE ('TEN ');
         11 : WRITE ('ELEVEN ');
         12 : WRITE ('TWELVE ');
         13 : WRITE ('THIRTEEN ');
         14 : WRITE ('FOURTEEN ');
         15 : WRITE ('FIFTEEN ');
         16 : WRITE ('SIXTEEN ');
         17 : WRITE ('SEVENTEEN ');
         18 : WRITE ('EIGHTEEN ');
         19 : WRITE ('NINETEEN ');
         20 : WRITE ('TWENTY ');
         30 : WRITE ('THIRTY ');
         40 : WRITE ('FOURTY ');
```

```
            50 : WRITE ('FIFTY ');
            60 : WRITE ('SIXTY ');
            70 : WRITE ('SEVENTY ');
            80 : WRITE ('EIGHTY ');
            90 : WRITE ('NINETY ')
         END
      END;

PROCEDURE TILL_HUNDRED (PRICE : INTEGER);
      BEGIN
        IF  PRICE < 10  THEN
           NUMBER_WORD (PRICE)
        ELSE
           IF  PRICE < 20  THEN
              TENS_WORD (PRICE)
           ELSE
            BEGIN
              TENS_WORD (PRICE - PRICE MOD 10);
              NUMBER_WORD (PRICE MOD 10)
            END
      END;

BEGIN  { main }
    WRITELN ('הכנס מחיר בדולרים ');
    READLN (CH, PRICE);
    IF  PRICE > 99  THEN        { אם המחיר הוא מספר בעל שלוש ספרות }
     BEGIN
       NUMBER_WORD (TRUNC (PRICE / 100));      { טיפול בספרת המאות }
       WRITE ('HUNDRED ')
     END;
    TILL_HUNDRED (TRUNC (PRICE) MOD 100);      { טיפול בשאר הספרות }
    IF  PRICE >= 1  THEN
       WRITE ('DOLLARS ');
    IF  (PRICE - TRUNC (PRICE)) * 100 > 0  THEN
     BEGIN                    { אם המחיר כולל סנטים - ספרות מימין לנקודה }
       TILL_HUNDRED (ROUND ((PRICE - TRUNC (PRICE)) * 100));
       WRITE ('CENTS ')
     END
END.
```

<u>הסבר:</u>

הפרוצדורות TENS_WORD ו-NUMBER_WORD מדפיסות מלים בהתאם לערך המספר שהן מקבלות.

הפרוצדורה TILL_HUNDRED נעזרת בשתי הפרוצדורות הנ"ל כדי לבטא במלים מספר דו-סיפרתי.

<u>הערה כללית:</u> יש לשים לב לאופן השימוש בפונקציה TURNC. פונקציה זו קוטעת את החלק הלא שלם של מספר ממשי ומחזירה מספר שלם. לכן, עשויה להיווצר שגיאה הנובעת מאופן ביצוע פעולות חשבון במספרים ממשיים במחשב.
לדוגמה נחשב: TURNC ((PRICE-TRUNC (PRICE))*100)

נניח שנתון ערך המשתנה: PRICE = 14.17
לפי הנוסחה, עלינו לקבל את הערך 17.
בפועל, יכול לחיות מצב שהמחשב יבצע את הפעולה 0.17*100 ונקבל 16.9999
בגלל תהליך ההכפלה במחשב (שגיאת עיגול, אשר שונה במחשבים שונים).
לאחר ביצוע הפונקציה TRUNC נקבל 16 ולא 17 כפי שציפינו.
כדי לפתור בעיה זו, אפשר להשתמש כאן בפונקציה ROUND (ראה שורה 68).

תוכנית 18

```
PROGRAM DIGITS (OUTPUT);
VAR
   D          : INTEGER;
   I, I2, I3 : REAL;

FUNCTION DIGIT_IN (DIGIT : INTEGER;
                       X : REAL     ) : BOOLEAN;
   VAR
      FOUND : BOOLEAN;
   BEGIN
      FOUND := FALSE;
      WHILE (X > 0) AND NOT FOUND  DO
       BEGIN
         IF  X - INT (X / 10) * 10 = DIGIT THEN
            FOUND := TRUE;
         X := INT (X / 10)
       END;
      DIGIT_IN := FOUND
   END;

BEGIN  { main }
   READLN (D);
   I := 1;
   REPEAT
      I2 := SQR (I);
      I3 := I2 * I;
      IF (DIGIT_IN (D, I) AND DIGIT_IN (D, I2) AND
          DIGIT_IN (D, I3))  THEN
         WRITELN (I:3:0, I2:10:0, I3:10:0);
      I := I + 1
   UNTIL  I > 100
END.
```

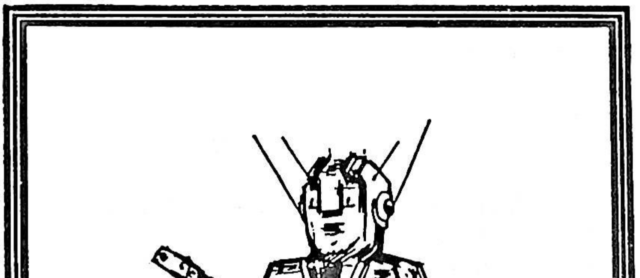
הסבר:

הפונקציה INT מבצעת קטיעה של החלק הלא שלם של מספר ממשי, בדומה לפונקציה TURNC, אך סוג התוצאה המוחזרת היא מספר ממשי ולא שלם. כלומר:

INT (17.382)=17.0 ; TRUNC (17.382)=17

השתמשנו פה ב-INT כי תהיה גלישה כאשר X > MAXINT

הפונקציה DIGIT_IN מקבלת סיפרה DIGIT ומספר שלם X (מוגדר ממשי כדי למנוע גלישה), ומחזירה TRUE אם הסיפרה מופיעה במספר, ו-FALSE - אם היא אינה מופיעה בו.



פרק 9
יישומים של פרוצדורות ופונקציות

שאלות

שאלה 2

הפרוצדורות המפורטות בהמשך מתאימות למסופים
הפועלים לפי הקוד המומלץ ע"י ANSI.

```
PROGRAM CONTROL (INPUT, OUTPUT);

{$P512}                    { להרצה ב-TURBO יש להוסיף את ההוראות הבאות }
{$G512}                    { ראה נספח ב' }

CONST
   UP    = 'A';            { תנועת הסמן בפרוצדורה CURSOR_POSITION }
   DOWN  = 'B';
   LEFT  = 'C';
   RIGHT = 'D';
   BOLD_MODE    = 1;       { אופן הכתיבה על המסך }
   UNDER_LINE   = 4;
   BLINK_MODE   = 5;
   REVERSE_MODE = 7;
   INVIS_MODE   = 8;

TYPE
   STRING_REC = RECORD
                   LENGTH : INTEGER;
                   STR    : ARRAY [1..65] OF CHAR
                END;
VAR
   S : STRING_REC;
   I : INTEGER;
```

פרק 9
יישומים של פרוצדורות ופונקציות

שאלות

שאלה 2

הפרוצדורות המפורטות בהמשך מתאימות למסופים
הפועלים לפי הקוד המומלץ ע"י ANSI.

```
PROGRAM CONTROL (INPUT, OUTPUT);

{$P512}                    { להרצה ב-TURBO יש להוסיף את ההוראות הבאות }
{$G512}                    { ראה נספח ב' }

CONST
   UP    = 'A';            { תנועת הסמן בפרוצדורה CURSOR_POSITION }
   DOWN  = 'B';
   LEFT  = 'C';
   RIGHT = 'D';
   BOLD_MODE    = 1;       { אופן הכתיבה על המסך }
   UNDER_LINE   = 4;
   BLINK_MODE   = 5;
   REVERSE_MODE = 7;
   INVIS_MODE   = 8;

TYPE
   STRING_REC = RECORD
                   LENGTH : INTEGER;
                   STR    : ARRAY [1..65] OF CHAR
                END;
VAR
   S : STRING_REC;
   I : INTEGER;
```

```
                                        { פרוצדורה להזזת הסמן ממקומו }

PROCEDURE CURSOR_POSITION (RELATIVE_STEP : INTEGER; {        גודל הצעד }
                           DIRECTION     : CHAR);   { כיוון ההתקדמות }
    BEGIN
      WRITE (CHR (27), '[', RELATIVE_STEP:1, DIRECTION)
    END;

PROCEDURE SAVE_CURSOR_POSITION;                     { שמירת מיקום הסמן }
    BEGIN
      WRITE (CHR (27), '[s')
    END;

PROCEDURE RESTORE_CURSOR_POSITION;         { החזרת הסמן למיקום, שנשמר ע"י }
    BEGIN                                  { הפרוצ' SAVE_CURSOR_POSITION. }
      WRITE (CHR (27), '[u')
    END;

PROCEDURE CLEAR_SCREEN;                                     { מחיקת מסך }
    BEGIN
      WRITE (CHR (27), '[2J')
    END;

PROCEDURE CLEAR_TO_EOLN;              { מחיקה ממקום הסמן ועד סוף השורה }
    BEGIN
      WRITE (CHR (27), '[K')
    END;


PROCEDURE LOCATION (X,                  { הזזת הסמן לשורה X ועמודה Y }
                    Y  : INTEGER);
    BEGIN
      WRITE (CHR (27), '[', X:1, ';', Y:1, 'H')
    END;

PROCEDURE NORMAL;                         { לאחר הפעלת הפרוצ' ייראה כל תו }
    BEGIN                                 { שירשם על המסך בצורתו הרגילה. }
      WRITE (CHR (27), '[0m')
    END;

PROCEDURE BOLD;          { לאחר הפעלת הפרוצ' יודגש כל תו שירשם על המסך }
    BEGIN
      WRITE (CHR (27), '[1m')
    END;

PROCEDURE UNDERLINE;                  { כל תו שירשם על המסך לאחר הפעלת }
    BEGIN                             { הפרוצ' יהיה מודגש בקו.          }
      WRITE (CHR (27), '[4m')
    END;
```

```
PROCEDURE BLINK;          { לאחר הפעלת הפרוצ' יהבהב כל תו שירשם על המסך }
    BEGIN
      WRITE (CHR (27), '[5m')
    END;

PROCEDURE REVERSE_VIDEO;                { כל תו שירשם על המסך לאחר הפעלת }
    BEGIN                               { הפרוצ' יוצג בהיפוך (inverse).   }
      WRITE (CHR (27), '[7m')
    END;

PROCEDURE INVISIBLE;                         { לאחר הפעלת הפרוצ' לא יראו }
    BEGIN                                    { התווים שירשמו על המסך.     }
      WRITE (CHR (27), '[8m')
    END;
                                       { הפרוצ' כותבת על המסך את מחרוזת S, }
PROCEDURE PRINT (X,                    { במיקום (X, Y): שורה X             }
                 Y,                    {                 עמודה Y            }
                 MODE : INTEGER;       { באופן הצגה אשר נקבע ע"י MODE.     }
             VAR S    : STRING_REC);
   VAR
      I : INTEGER;
    BEGIN
      LOCATION (X, Y);
      WRITE (CHR (27), '[', MODE:1, 'm');
      WITH  S  DO
         FOR  I := 1  TO  LENGTH  DO
            WRITE (STR [I]);
      NORMAL
    END;

BEGIN { main }

END.
```

הערה:

א. במחשבי IBM-PC עם מערכת הפעלה DOS, יש לכלול בקובץ CONFIG.SYS את ההצבה הבאה:

device = [d:][path] ansi.sys

יש לדאוג לכך, שהקובץ ansi.sys אכן נמצא ב-path המתאים.

ב. אפשר לשלב בין אופני ההצגה השונים. לדוגמה: אפשר להפעיל את הפרוצ' BOLD וגם את הפרוצ' UNDERLINE בכדי שהתווים יוצגו בהדגשת הארה ועם קו מתחתם.

תוכניות

תוכנית 1

הערה: ראה בנספח לטורבו את ההסבר לפונקציה RANDOM.

```
PROGRAM RANDOM_NUMBERS (OUTPUT);
TYPE
   RESULTS = ARRAY [1..6, 1..6] OF INTEGER;  { מערך הסופר את התוצאות }
                                             { של זריקת הקוביות.     }
VAR
   RESULT  : RESULTS;
   I, J,
   DISH1,
   DICE_1, DICE_2 : INTEGER;

PROCEDURE OTAKIM (C : CHAR;          { הפרוצדורה מדפיסה N פעמים את התו C }
                  N : INTEGER);
  VAR
     I : INTEGER;
  BEGIN
     FOR  I := 1  TO  N  DO
        WRITE (C);
     WRITELN
  END;     { otakim }

BEGIN { main }
  FOR  I:= 1  TO  6  DO
     FOR  J := 1  TO  6  DO
        RESULT [I, J] := 0;                           { איפוס מערך התוצאות }
  FOR  I := 1  TO  1000  DO
   BEGIN
     DICE_1 := 1 + RANDOM (6); { בחירת מספר אקראי מ-1 עד 6, קוביה 1 }
     DICE_2 := 1 + RANDOM (6); { בחירת מספר אקראי מ-1 עד 6, קוביה 2 }
     RESULT [DICE_1, DICE_2] := RESULT [DICE_1, DICE_2] + 1
                { במטריצת התוצאה סופרים את התוצאות של זריקות הקוביות }
   END;
  FOR  I := 2  TO  6  DO { איחוד תוצאות דומות, למשל: (5,4) ו- (4,5) }
     FOR  J := 1  TO  I - 1  DO
        RESULT [I, J] := RESULT [I, J] + RESULT [J, I];
  FOR  I:= 1  TO  6  DO     { הדפסת מטריצת התוצאות כהיסטוגרמה אופקית }
     FOR  J := 1  TO  I  DO
      BEGIN
        WRITE(I:2, J:2, RESULT [I, J]:4, '  ');
        OTAKIM ('*', RESULT [I, J])
      END
END.
```

הסבר:

	1	2	3	4	5	6
1						
2	X					
3	X	X				
4	X	X	X			
5	X	X	X	X		
6	X	X	X	X	X	

במקומות המסומנים ב-X נספרות התוצאות הכפולות.

בהדפסת מטריצת התוצאות עוברים הן על האיברים המסומנים ב-X והן על איברי האלכסון.

תוכנית 2 א'

```
PROGRAM CRAPS (INPUT, OUTPUT);
VAR
   FIRST_DICE,
   SUM,
   MONEY,
   BET            : INTEGER;

FUNCTION DICE_SUM : INTEGER;              { הפונקציה מדמה הטלת שתי קוביות }
   VAR                                    { ומחזירה את סכום הערכים שהתקבלו }
      SUM : INTEGER;
   BEGIN
      SUM := 2 + RANDOM(6) + RANDOM(6);
      WRITELN (SUM:2, ' : סכום הקוביות');
      DICE_SUM := SUM
   END;

BEGIN {main}
   MONEY := 0;
   WRITELN ('[ להפסקת המשחק - 0 ] הכנס את סכום ההימור');
   READLN (BET);
   WHILE BET > 0 DO
    BEGIN
      FIRST_DICE := DICE_SUM;                      { סכום ההטלה הראשונה }
      IF  (FIRST_DICE = 7) OR (FIRST_DICE = 11) THEN
```

```
        MONEY := MONEY + BET                              { זכייה מיידית }
      ELSE
        IF  (FIRST_DICE = 2) OR (FIRST_DICE = 3) OR       { הפסד מיידי }
         (FIRST_DICE = 12) THEN
          MONEY := MONEY - BET
        ELSE
          REPEAT
            SUM := DICE_SUM;                              { הטלה נוספת }
            IF  SUM = FIRST_DICE  THEN          { אם הטלה זו שווה להטלה }
              MONEY := MONEY + BET              { ראשונה אז זוהי זכייה. }
            ELSE
              IF  SUM = 7  THEN  { אם בהטלה זו יצא 7 אז זהו הפסד }
                MONEY := MONEY - BET
          UNTIL (SUM = FIRST_DICE) OR (SUM = 7); { הלולאה נמשכת עד }
                                          { שאחד התנאים לעיל מתמלא. }
      WRITELN (MONEY:2, ' : מאזן הזכיות');
      WRITELN;
      WRITELN ('  הכנס את סכום ההימור');
      READLN (BET)
    END   { while bet > 0 }
END.
```

תוכנית 2 ב'

```
PROGRAM CRAPS (INPUT, OUTPUT);
VAR
   I,
   FIRST_DICE,
   SUM,
   WIN           : INTEGER;

FUNCTION DICE_SUM : INTEGER;
    BEGIN
      DICE_SUM := 2 + RANDOM (6) + RANDOM (6)
    END;

BEGIN {main}
   WIN := 0;
   FOR  I := 1  TO  1000  DO
    BEGIN
      FIRST_DICE := DICE_SUM;
      IF  (FIRST_DICE = 7) OR (FIRST_DICE = 11) THEN
         WIN := WIN + 1
      ELSE
         IF  (FIRST_DICE <> 2) AND (FIRST_DICE <> 3) AND
          (FIRST_DICE <> 12) THEN
            REPEAT
               SUM := DICE_SUM;
               IF  SUM = FIRST_DICE  THEN
```

```
                WIN := WIN + 1
          UNTIL (SUM = FIRST_DICE) OR (SUM = 7)
    END;    { for i ...}
   WRITELN (WIN / 1000:6:4, 'ההסתברות שהשחקן יזכה היא ')
END.
```

<u>הסבר:</u>

תוכנית זו מאד דומה לתוכנית 2א'. בתוכנית זו משחקים 1000 פעמים ורק הזכיות נספרות, לכן הסיכוי הוא מספר הזכיות חלקי 1000.

<u>הערה:</u> בתוכנית 2ב' אין צורך לשאול את השאלה SUM = 7 שבתוך לולאת ה-REPEAT, כי זהו מצב הפסד, והרי אין אנו סופרים מצבים אלה. בכל זאת, יש לשאול את השאלה במשפט התנאי של לולאת ה-REPEAT כדי לצאת מהלולאה.

תוכנית 3

```
PROGRAM RANDOM_INTEGRAL (OUTPUT);
CONST
   POINTS_NUMBER = 5000;
VAR
   I,
   INSIDE       : INTEGER;
   LINE_INSIDE : BOOLEAN;

FUNCTION Y_SMALLER_THAN_SQR_X : BOOLEAN;
   VAR
      SQR_X,
      Y        :REAL;
   BEGIN
      SQR_X := SQR (RANDOM (MAXINT) / (MAXINT - 1));
      Y := RANDOM (MAXINT) / (MAXINT - 1);
      LINE_INSIDE := LINE_INSIDE  XOR  (Y = SQR_X);   { בכל פעם ש-Y   }
                                                      { שווה ל-SQR_X. }
                         LINE_INSIDE = NOT LINE_INSIDE.
                          { כאשר נקודה נמצאת על הקו, המשתנה
                           LINE_INSIDE קובע אם היא נחשבת בתוך התחום. }
      Y_SMALLER_THAN_SQR_X :=
           (Y < SQR_X) OR (LINE_INSIDE AND (Y = SQR_X))
   END;

BEGIN    {main}
   INSIDE := 0;
   LINE_INSIDE := TRUE;
   FOR I := 1  TO  POINTS_NUMBER  DO
      IF  Y_SMALLER_THAN_SQR_X  THEN
         INSIDE := INSIDE + 1;
   WRITELN (INSIDE / POINTS_NUMBER:8:6, 'תוצאת האינטגרציה  ')
END.
```

תוכנית 4

```
PROGRAM BLACK_JACK (INPUT, OUTPUT);
TYPE
   CARDS = 1..11;                                         { תחום ערכי הקלפים }
VAR
   PACK   : ARRAY [1..52] OF CARDS;                       { חפיסת הקלפים }
   MONEY,
   BET,
   I,
   J      : INTEGER;
   ANSWER : CHAR;

FUNCTION GAME : BOOLEAN;
   VAR
      COMPUTER_HAND,
      MAN_HAND,
      I,
      J,
      ADD_BET : INTEGER;
      COMP_AS,
      MAN_AS,
      COMP_WIN,
      MAN_WIN : BOOLEAN;
                                          { פונקציה המחזירה את ערך הקלפים }
   FUNCTION EVAL (X  : INTEGER;
                  AS : BOOLEAN) : INTEGER;
      BEGIN
         IF  AS AND (X + 10 <= 21) AND (X + 10 >= 17) THEN
            {אם יש אס וסכום הקלפים בין 17 ל-21 אז ערכו של אס שווה 11}
            EVAL := X + 10
         ELSE                                       { אחרת ערכו של אס שווה 1 }
            EVAL := X
      END;  { eval }

   PROCEDURE STATUS (VAR HAND      : INTEGER;         { ניתוח מצב במשחק }
                     VAR AS,
                         HAND_WIN,
                         OTHER_WIN : BOOLEAN);
      BEGIN
         IF  EVAL (HAND, AS) > 21  THEN
               OTHER_WIN := TRUE                     { השחקן השני ניצח }
            ELSE
               IF  EVAL (HAND, AS) = 21  THEN
                  HAND_WIN := TRUE  {השחקן שערך הקלפים שלו 21 - ניצח}
      END;    { status }
```

```
PROCEDURE TRIFA;                          { פרוצדורה לטריפת הקלפים שבחפיסה }
   VAR
      I,
      J    : INTEGER;
      TEMP : CARDS;
   BEGIN
      FOR  J := 52 DOWNTO  1  DO
       BEGIN
         I := 1 + RANDOM (52);
         TEMP := PACK [I];
         PACK [I] := PACK [J];
         PACK [J] := TEMP
       END
   END;  { trifa }

BEGIN  { game }
   COMPUTER_HAND := 0;          { ערך הקלפים שברשות המחשב }
   MAN_HAND      := 0;          { ערך הקלפים שבידי השחקן }
   COMP_AS       := FALSE;      { מציין אם למחשב יש אס }
   MAN_AS        := FALSE;      { מציין אם לשחקן יש אס }
   COMP_WIN      := FALSE;      { מחשב ניצח }
   MAN_WIN       := FALSE;      { שחקן ניצח }
   TRIFA;                       { טריפת החפיסה }
   ANSWER := 'Y';               { ערך התחלתי למשיכת קלף ראשון }
   I      := 52;                { מספר הקלף בראש החפיסה }
   BET    := 0;                 { סכום ההימור }
   WHILE NOT MAN_WIN AND
         NOT COMP_WIN AND
         ((ANSWER = 'Y') OR
         (EVAL (COMPUTER_HAND, COMP_AS) < 17) OR
      (EVAL(COMPUTER_HAND, COMP_AS) < EVAL(MAN_HAND, MAN_AS)))  DO
    BEGIN        { לולאה כל זמן ש: - מחשב לא ניצח,
                 {                 - שחקן לא ניצח,
                                   - התשובה למשוך היא 'כן',
                                   - ערך הקלפים של המחשב קטן מ-17
                                     או קטן מערך הקלפים שבידי השחקן. }
     IF  (EVAL (COMPUTER_HAND, COMP_AS) < 17) OR
      (EVAL(COMPUTER_HAND, COMP_AS) < EVAL(MAN_HAND, MAN_AS)) THEN
      BEGIN                      { ערך הקלפים שבידי המחשב קטן מ-17 }
                                 { או קטן מערך הקלפים שבידי השחקן. }
        COMPUTER_HAND := COMPUTER_HAND + PACK [I];        { מחשב }
                                                          { לוקח קלף. }
        IF  PACK [I] = 1  THEN                            { אם "אס" }
           COMP_AS := TRUE;
           WRITE ('THE CARD I GET IS ', PACK [I]:2);
        I := I - 1                         { מצביע לקלף הבא בחפיסה }
      END
     ELSE
        WRITE (' ':20);
     STATUS (COMPUTER_HAND, COMP_AS, COMP_WIN, MAN_WIN);
                                                  { הערכת מצב המשחק }
```

```
        IF  NOT (COMP_WIN OR MAN_WIN)  THEN
         BEGIN                                            { אם עדיין אין נצחון }
           WRITE ('ANOTHER CARD? ':17);
           READ (ANSWER);
           IF  ANSWER = 'Y'  THEN                          { שחקן מבקש עוד קלף }
            BEGIN
              WRITE('  YOUR BET? ');
              READ (ADD_BET);                                   { הימור }
              BET := BET + ADD_BET;
              MAN_HAND := MAN_HAND + PACK [I];                { שחקן לוקח קלף }
              IF  PACK [I] = 1  THEN                           { אם "אס" }
                 MAN_AS := TRUE;
              WRITELN ('THE CARD YOU GET IS ':24, PACK [I]:2);
              I := I - 1                            { מצביע לקלף הבא בחפיסה }
            END
           ELSE
              WRITELN;
           STATUS (MAN_HAND, MAN_AS, MAN_WIN, COMP_WIN)
                                                         { הערכת מצב המשחק }
         END    { if }
      END;   { while }
     WRITELN;
     WRITE ('I HAVE ', EVAL (COMPUTER_HAND, COMP_AS):5,  { מצב סופי }
        ' YOU HAVE ', EVAL (MAN_HAND, MAN_AS):5);
     IF  COMPUTER_HAND > 21  THEN                        { ערך הקלפים שבידי }
                                                        { המחשב גדול מ-21. }
        WRITELN ('   I BLOW UP YOU WIN !')
     ELSE
        IF  MAN_HAND > 21  THEN  { ערך הקלפים שבידי השחקן גדול מ-21 }
           WRITELN ('   YOU BLOW UP I WIN !')
        ELSE
           IF  MAN_WIN  THEN
              WRITELN ('   YOU WIN !')
           ELSE
              WRITELN ('   I WIN !');
     GAME := MAN_WIN OR NOT COMP_WIN          { הפונקציה מחזירה TRUE אם }
                                              { שחקן ניצח או מחשב הפסיד. }
   END; { game }

BEGIN  { main }
   FOR  I := 1  TO  10  DO                             { בניית חפיסת הקלפים }
      FOR  J := 0  TO  3  DO
         PACK [I + J * 10] := I;
   FOR  I := 41  TO  52  DO
      PACK [I] := 10;
   MONEY := 0;
   REPEAT
      IF  GAME  THEN
         MONEY := MONEY + BET
      ELSE
         MONEY := MONEY - BET;
```

```
      WRITELN ('MONEY = ', MONEY:1);
      WRITE ('ANOTHER GAME ? ');
      READLN (ANSWER)
   UNTIL ANSWER <> 'Y'
END. { main }
```

תוכנית 5

```
FUNCTION BIGGER (FUNCTION F (Y:REAL) : REAL;
                 FUNCTION G (V:REAL) : REAL;
                 X : REAL                    ) : REAL;
BEGIN
   IF  F (X) > G (X)  THEN
    BEGIN
      WRITELN ('FIRST FUNCTION IS BIGGER');
      BIGGER := F (X)
    END
   ELSE
    BEGIN
      WRITELN ('SECOND FUNCTION IS BIGGER');
      BIGGER := G (X)
    END
 END;
```

<u>הסבר:</u>

פונקציה BIGGER מקבלת שתי פונקציות ממשיות F ו-G ומשתנה ממשי X,
ומחזירה ערך ממשי.
כל אחת מהפונקציות (F ,G) היא פונקציה ממשית המקבלת ערך ממשי ומחזירה
ערך ממשי.
הערך הממשי שתחזיר הפונקציה BIGGER יהיה הערך הממשי הגדול מבין שני
ערכים: ערך אחד שתחזיר הפונקציה F עם משתנה ממשי X,
וערך שני שתחזיר הפונקציה G עם משתנה ממשי X.

<u>הערה:</u> תוכנית זו נכתבה במהדר פסקל על מחשב VAX/VMS.
תוכנית זו לא תפעל בגירסת הטורבו פסקל על מחשב IBM-PC.

קיימים מהדרים של שפת פסקל אשר אינם מאפשרים לפונקציות סטנדרטיות (כמו
(SIN (X) להיות פרמטר של פונקציה אחרת (ראה הסבר בספר "פסקל" עמוד
202).

קיימים מהדרים הדורשים שפרמטר שהוא פונ'/פרוצ', המופיע בהגדרת
פונ'/פרוצ' אחרת, יפרט את כל המשתנים של אותה פונ'/פרוצ'. לדוגמה, ראה
תוכניות 5 ו-6.

בפסקל סטנדרטי כל המשתנים של פונ'/פרוצ', שהיא פרמטר של פונ'/פרוצ'
אחרת, חייבים להיות פרמטרים ערכיים (לא VAR).

תוכנית 6

```
PROGRAM P (INPUT,OUTPUT);

FUNCTION INTEGRAL (FUNCTION F (X : REAL) : REAL;
                   A, B, : RAEL
                   N     : INTEGER                    ) : REAL;
  VAR
     RESULT,
     H,
     SIGN : REAL;
     I    : INTEGER;
  BEGIN
     H := ABS (A - B) / N;                        { רוחב הטרפז }
     SIGN := (B - A) / ABS (B - A);               { סימן האינטגרל }
     RESULT := 0.5 * H * (F (A) + F (B));
     IF  B < A  THEN
        A := B;                                   { התחלה מגבול תחתון }
     FOR  I := 1  TO  N - 1  DO
        RESULT := RESULT + H * F (A + I * H);
     INTEGRAL := SIGN * RESULT
  END;

BEGIN { main }
   WRITELN (INTEGRAL (SIN, 0, 1, 100));
           { אינטגרל של סינוס בין 0 ל-1, חלוקה ל-100 יחידות חישוב }
      :
      :
   WRITELN (INTEGRAL (SQR, 100, 200, 1000));
      :
      :
   WRITELN (INTEGRAL (SQRT, 0, 3, 300))
      :
END.
```

<u>הסבר:</u>

הפונקציה INTEGRAL מקבלת כפרמטרים:
א. פונקציה F "ממשית".
ב. משתנים ממשים A ו-B המציינים את גבולות האינטגרל.
ג. משתנה שלם N המציין את מספר החלוקות לטרפזים.
היא מחזירה ערך ממשי.

הפונקציה F מקבלת ערך ממשי ומחזירה ערך ממשי.

ראה הערות לתוכנית מספר 5 לגבי פונ'/פרוצ' העוברת כפרמטר לפונ'/פרוצ' אחרת.

פרק 10
רשומות

שאלות

שאלות 1 ו-2

(שאלה 2 מתחילה בשורה 31).

```
(1)  TYPE
(2)     CHR_STRING = PACKED ARRAY [1..15] OF CHAR;
(3)     FULL_NAME  = RECORD                 { שם מלא: שם פרטי ושם משפחה }
(4)                     FIRST_NAME,
(5)                     FAMILY_NAME : CHR_STRING
(6)                  END;
(7)     DATE = RECORD                       { תאריך לידה: יום, חודש, שנה }
(8)               DAY   : 1..31;
(9)               MONTH : 1..12;
(10)              YEAR  : 1900..2000
(11)           END;
(12)    STREET_REC = RECORD                             { רחוב: שם ומספר }
(13)                    NAME   : CHR_STRING;
(14)                    NUMBER : 1..MAXINT
(15)                 END;
(16)    ADDRESS_REC = RECORD                { כתובת: עיר, מיקוד ורחוב }
(17)                     CITY,
(18)                     ZIP_CODE : CHR_STRING;
(19)                     STREET   : STREET_REC
(20)                  END;
```

```
(21)     NETUNIM = RECORD
(22)                 NAME       : FULL_NAME;            { שם }
(23)                 ID         : REAL;      { מספר זהות גדול מ-MAXINT }
(24)                 BORN_DATE  : DATE;                 { תאריך לידה }
(25)                 CHILDREN   : 1..16;                { מספר ילדים }
(26)                 STATUS     : (RAVAK, NASUI, ALMAN,  { מצב משפחתי }
(27)                               , GARUSH, NASUI_2);
(28)                 ADDRESS    : ADDRESS_REC;          { כתובת }
(29)                 SEX        : (MALE, FEMALE)        { מין }
(30)              END;
(31) VAR
(32)     PRATIM : NETUNIM;  { הגדרת המשתנה PRATIM כרשומה מסוג NETUNIM }
(33)     N      : INTEGER;
(34) BEGIN
(35)     WITH  PRATIM  DO
(36)      BEGIN         { מכאן ועד שורה 67 מתייחסים לשדות ברשומה PRATIM }
(37)        WITH  NAME  DO
(38)         BEGIN              { שתי השורות הבאות מתייחסות ל-PRATIM.NAME }
(39)           READ (FIRST_NAME);
(40)           READ (FAMILY_NAME)
(41)         END;
(42)        READ (ID);
(43)        WITH  BORN_DATE  DO
(44)         BEGIN   { שלוש השורות הבאות מתייחסות ל- PRATIM.BORN_DATE }
(45)           READ (DAY);
(46)           READ (MONTH);
(47)           READ (YEAR)
(48)         END;
(49)        READ (CHILDREN);
(50)        READ (N);
(51)        CASE  N  OF                 { שימוש ב-CASE, כי לא ניתן לקרוא }
(52)           1 : STATUS := RAVAK;  { בצורה ישירה את הערכים הדרושים. }
(53)           2 : STATUS := NASUI;
(54)           3 : STATUS := ALMAN;
(55)           4 : STATUS := GARUSH;
(56)           5 : STATUS := NASUI_2
(57)        END;
(58)        WITH  ADDRESS, STREET  DO  { ארבע השורות הבאות מתיחסות ל: }
(59)         BEGIN        { PRATIM.ADDRESS, או ל-PRATIM.ADDRESS.STREET }
(60)           READ (CITY);
(61)           READ (NAME);
(62)           READ (NUMBER);
(63)           READ (ZIP_CODE)
(64)         END;
(65)        READ (N);
(66)        CASE  N  OF                 { שימוש ב-CASE, כי לא ניתן לקרוא }
(67)           1 : SEX := MALE;         { בצורה ישירה את הערכים הדרושים. }
(68)           2 : SEX := FEMALE
(69)        END
(70)     END
(71) END.
```

שאלה 4

```
CONST
   MAX = 80;
TYPE
   CHR_NUMBER = 1..MAX;   { מצביע על מיקום תווים במערך מסוג STR }
   STR        = PACKED ARRAY [1..MAX] OF CHAR;   { מערך של תווים }
   CHR_STR    = RECORD
                   LENGTH : CHR_NUMBER;     { מספר תווים במערך }
                   TEXT   : STR             { מערך }
                END;

{ הפונקציה CONNECT מקבלת: - מחרוזת משתנה M,
                         - מחרוזת MH,
                         - משתנה FROM המצביע על המקום המיועד
                           להוספה של מחרוזת MH למחרוזת M. }

FUNCTION CONNECT (VAR M    : CHR_STR;
                      MH   : CHR_STR;
                      FROM : CHR_NUMBER) : INTEGER;
   VAR
      I, J : CHR_NUMBER;
      CODE : INTEGER;                { הערך שיוחזר על ידי הפונקציה }
   BEGIN
      CODE := 0;
      IF  FROM > M.LENGTH + 1  THEN                 { הוספת רווחים }
       BEGIN
         CODE := 1;
         FOR  I := M.LENGTH + 1  TO  FROM - 1  DO
            M.TEXT [I] := ' ';
         M.LENGTH := FROM - 1
       END;
      IF  M.LENGTH + MH.LENGTH <= MAX  THEN       {   הוספה רגילה }
         M.LENGTH := M.LENGTH + MH.LENGTH
      ELSE
          BEGIN
            M.LENGTH := MAX;
            CODE := CODE + 2
          END;
      FOR  I := M.LENGTH  DOWNTO  FROM + MH.LENGTH  DO  { פתיחת
                           רווח במחרוזת M לשם הכנסת מחרוזת MH }
               M.TEXT [I] := M.TEXT [I - MH.LENGTH];
      IF  M.LENGTH >= FROM + MH.LENGTH  THEN
         J := FROM + MH.LENGTH - 1
      ELSE
         J := M.LENGTH;
      FOR  I := FROM  TO  J  DO { הכנסת מחרוזת MH למקום שפינינו
         M.TEXT [I] := MH.TEXT [I - FROM + 1];      במחרוזת M. }
      CONNECT := CODE
   END;
```

שאלה 5 א'

```
(1)  CONST
(2)     MAX = 80;
(3)  TYPE
(4)     CHR_NUMBER = 1..MAX;
(5)     STR        = PACKED ARRAY [1..MAX] OF CHAR;
(6)     CHR_STR    = RECORD
(7)                     LENGTH : CHR_NUMBER;
(8)                     TEXT   : STR
(9)                  END;
```

{ הפרוצדורה CHIBUR מחזירה במשתנה M מחרוזת,
הנוצרת מחיבור של מחרוזת MH בסופה של מחרוזת M. }

```
(10) PROCEDURE CHIBUR (VAR M  : CHR_STR;
(11)                       MH : CHR_STR);
(12)    VAR
(13)       I, J : CHR_NUMBER;
```

{ הפונקציה MIN מחזירה את הערך המינימלי
מבין שני הפרמטרים שהיא מקבלת. }

```
(14)    FUNCTION MIN (A, B  : CHR_NUMBER) : CHR_NUMBER;
(15)    BEGIN
(16)       IF  A < B  THEN
(17)          MIN := A
(18)       ELSE
(19)          MIN := B
(20)    END;   { min }

(21)    BEGIN   { chibur }
(22)      FOR I := M.LENGTH + 1 TO  MIN (M.LENGTH + MH.LENGTH, MAX) DO
(23)            M.TEXT [I] := MH.TEXT [I - M.LENGTH];
(24)       M.LENGTH := MIN (M.LENGTH + MH.LENGTH, MAX)
(25)    END;      { chibur }
```

הסבר:

22-23 התוספה של המחרוזת MH הינה מהמקום הראשון שלאחר מחרוזת M ועד לערך הקטן יותר מבין סכום אורכי המחרוזות (MH ו- M) לבין הערך MAX, זאת, כדי שסכום האיברים ב-M לאחר החיבור לא יעלה על 80 תווים.

24 אורך המחרוזת M מעודכן לערכו החדש.

שאלה 6

```
(1)  CONST
(2)     MAX = 80;
(3)  TYPE
(4)     CHR_NUMBER = 1..MAX;
(5)     STR        = PACKED ARRAY [1..MAX] OF CHAR;
(6)     CHR_STR    = RECORD
(7)                    LENGTH : CHR_NUMBER;
(8)                    TEXT   : STR
(9)                  END;

(10) PROCEDURE HOTZAA (VAR M,
(11)                       ML   : CHR_STR;
(12)                       FROM,
(13)                       LEN  : CHR_NUMBER);
(14)    VAR
(15)       I, J : CHR_NUMBER;

(16)    FUNCTION MIN (A, B  : CHR_NUMBER) : CHR_NUMBER;
(17)    BEGIN
(18)       IF  A < B  THEN
(19)          MIN := A
(20)       ELSE
(21)          MIN := B
(22)    END;

(23)    BEGIN { hotzaa }
(24)       J := MIN (FROM + LEN - 1, MAX);
(25)       FOR  I := FROM  TO  J  DO
(26)          ML.TEXT [I - FROM + 1] := M.TEXT [I];
(27)       ML.LENGTH := J - FROM + 1;
(28)       J := MIN (FROM + LEN, M.LENGTH + 1);
(29)       FOR  I := 0  TO  M.LENGTH - J  DO
(30)          M.TEXT [I + FROM] := M.TEXT [J + I];
(31)       M.LENGTH := M.LENGTH - J + FROM
(32)    END;     { hotzaa }
```

הסבר:

24 משתנה J הוא האינדקס של סוף המחרוזת שיש להוציא.
25-26 העתקה של המחרוזת שיש להוציא, אל המחרוזת ML.
27 אורך המחרוזת שהוצאנו.
29-30 מחיקת המחרוזת על ידי הזזת התווים שאחריה למקום המתאים.
31 עדכון האורך הכללי של המחרוזת.

שאלה 7 ב'

<u>הערה:</u> בעיה 7א' הינה מקרה פרטי של 7ב', כאשר המשתנה CONTINUE הוא FALSE. להלן הפתרון של שאלה 7 ב'.

```
CONST
   MAX = 80;
TYPE
   CHR_NUMBER = 1..MAX;
   STR        = PACKED ARRAY [1..MAX] OF CHAR;
   CHR_STR    = RECORD
                   LENGTH : CHR_NUMBER;
                   TEXT   : STR
                END;

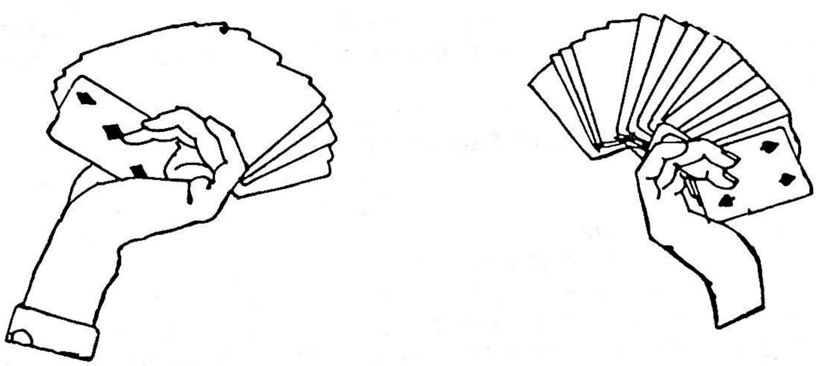
PROCEDURE HACHLAFA (VAR M          : CHR_STR;
                        REPLACE,
                        EXAMPLE    : CHR_STR;
                        FROM       : CHR_NUMBER;
                    VAR FOUND      : INTEGER;
                        CONTINUE   : BOOLEAN);
   VAR
      I, J  : CHR_NUMBER;
      EQUAL : BOOLEAN;

PROCEDURE DELETE_STR (VAR M    : CHR_STR;
                          FROM,
                          LEN  : CHR_NUMBER);
   VAR
      I : CHR_NUMBER;

   BEGIN
      FOR  I := FROM  TO  M.LENGTH - LEN   DO         {        מחיקה }
         M.TEXT [I] := M.TEXT [I + LEN];
      M.LENGTH := M.LENGTH - LEN                      { עדכון אורך }
   END; { delete_str }

PROCEDURE CONNECT (VAR M    : CHR_STR;
                       MH   : CHR_STR;
                       FROM : CHR_NUMBER);
   VAR
      I : CHR_NUMBER;
   BEGIN
      M.LENGTH := M.LENGTH + MH.LENGTH;               { עדכון אורך }
      FOR I := M.LENGTH DOWNTO FROM + MH.LENGTH DO { הזזה קדימה }
         M.TEXT [I] := M.TEXT [I - MH.LENGTH];   { לפינוי מקום. }
      FOR I := FROM TO FROM + MH.LENGTH - 1 DO        { הכנסה של }
         M.TEXT [I] := MH.TEXT [I - FROM + 1] { המחרוזת למקומה. }
   END; { connect }
```

```
BEGIN { hachlafa }
   FOUND := 0;
   I     := FROM - 1;
   WHILE  I < M.LENGTH - EXAMPLE.LENGTH  DO
    BEGIN
      EQUAL := TRUE;                    { מצב התחלתי - יש שיוויון }
      J := 1;
      WHILE  (J <= EXAMPLE.LENGTH) AND EQUAL  DO     { כל זמן }
       BEGIN   { שיש שיוויון ולא נבדקו כל המחרוזות ב-EXAMPLE }
         IF  M.TEXT [I + J] <> EXAMPLE.TEXT [J]  THEN
            EQUAL := FALSE;                        { אין שיוויון }
         J := J + 1
       END;
      IF  EQUAL  THEN
       BEGIN
         FOUND := 1 + I;             { המקום שבו נמצאת המחרוזת }
         IF  M.LENGTH - EXAMPLE.LENGTH + REPLACE.LENGTH > MAX
            THEN FOUND := -1    { החלפה לא מתבצעת מחוסר מקום }
         ELSE
             BEGIN
               DELETE_STR (M, FOUND, EXAMPLE.LENGTH); {מחיקה}
               CONNECT (M, REPLACE, FOUND);           { הוספה }
               I := I + REPLACE.LENGTH - EXAMPLE.LENGTH
             END;
         IF  NOT CONTINUE  THEN
            I := M.LENGTH { יציאה מלולאת ה-WHILE ומהפרוצדורה }
       END;   { if equal }
      I := I + 1
    END   { while }
END;    { hachlafa }
```



"משחק קלפים
שיטת מונטה-קרלו"

תוכניות

תוכנית 1

```
PROGRAM QUERY (INPUT, OUTPUT);
CONST
   MAX_STUDENTS = 40;
TYPE
   STUDENT_REC = RECORD
                    NAME     : PACKED ARRAY [1..10] OF CHAR;
                    FAMILY   : PACKED ARRAY [1..15] OF CHAR;
                    CHILDREN : 1..20;
                    FATHER,
                    MOTHER   : 0..20;
                    MARK     : 0..100
                  END;
VAR
   STUDENT : ARRAY [1..MAX_STUDENTS] OF STUDENT_REC;
   LOW_5_AVERAGE,
   HIGH_5_AVERAGE,
   LESS_THAN_10_COUNTER,
   MORE_THAN_10_COUNTER,
   I, J    : INTEGER;

                          { קריאת רשומת סטודנט. הפונ' תחזיר FALSE אם
                              זהו סוף הקלט, אחרת - היא תחזיר TRUE. }

FUNCTION READ_DATA (I : INTEGER) : BOOLEAN;
   BEGIN
      WITH  STUDENT [I]  DO
         READLN (NAME, FAMILY, CHILDREN, FATHER, MOTHER, MARK);
    READ_DATA := NOT (STUDENT [I].NAME [1] = '.')
   END;   { read_data }

                                          { מיון רשומות הסטודנטים }
PROCEDURE SORT;              { לפי מספר ילדים במשפחה - מיון בועות }
   VAR
      STD, J, L : INTEGER;
      X         : STUDENT_REC;
   BEGIN
     J := I;
     WHILE  J > 0  DO
      BEGIN
       L := 0;
       FOR  STD := 1  TO  J - 1  DO
         IF  STUDENT [STD].CHILDREN > STUDENT [STD + 1].CHILDREN
           THEN BEGIN
                  X := STUDENT [STD];
                  STUDENT [STD] := STUDENT [STD + 1];
```

```
                    STUDENT [STD + 1] := X;
                    L := STD
                  END;
          J := L
       END   { while j }
    END;   { sort }

BEGIN { main }
    I := 0;
    WRITELN ('                                              שנות לימוד');
    WRITELN
       ('שם פרטי     שם משפחה     מס. ילדים     אב     אם     ציון');
    WHILE  READ_DATA (I + 1)  DO                      { קריאת נתונים }
       I := I + 1;
    SORT;                                                     { מיון }
    LOW_5_AVERAGE         := 0;
    HIGH_5_AVERAGE        := 0;
    LESS_THAN_10_COUNTER  := 0;
    MORE_THAN_10_COUNTER  := 0;
    WRITELN ('שמות 5 הסטודנטים מהמשפחות הקטנות':60);
    WRITELN ('שם פרטי':8, 'שם משפחה':15,
             'סכום שנות לימוד של ההורים':28);
    FOR  J := 1  TO  5  DO     { משפחות עם מספר ילדים הקטן ביותר }
       WITH  STUDENT [J]  DO
        BEGIN
          WRITELN (NAME, FAMILY, FATHER + MOTHER:15);
          LOW_5_AVERAGE := LOW_5_AVERAGE + MARK
        END;
    WRITELN (LOW_5_AVERAGE / 5:6:2,
             'ממוצע ציוני הסטודנטים בקבוצה זו :');
    WRITELN ('שמות 5 הסטודנטים מהמשפחות הגדולות':60);
    WRITELN ('שם פרטי':8, 'שם משפחה':15,
             'סכום שנות לימוד של ההורים':28);
    FOR J := I - 4  TO  I  DO  { משפחות עם מספר ילדים הגדול ביותר }
       WITH  STUDENT [J]  DO
        BEGIN
          WRITELN (NAME, FAMILY, FATHER + MOTHER:15);
          HIGH_5_AVERAGE := HIGH_5_AVERAGE + MARK
        END;
    WRITELN (HIGH_5_AVERAGE / 5:6:2,
             'ממוצע ציוני הסטודנטים בקבוצה זו :');
    FOR  J := 1  TO  I  DO
       WITH  STUDENT [J]  DO
        BEGIN
          IF  ((MOTHER > 10) OR (FATHER > 10)) AND (MARK > 90)
              THEN MORE_THAN_10_COUNTER := MORE_THAN_10_COUNTER + 1;
          IF  ((MOTHER < 10) OR (FATHER < 10)) AND (MARK > 90) THEN
               LESS_THAN_10_COUNTER := LESS_THAN_10_COUNTER + 1
        END;
```

```
   WRITELN
 ('סטודנטים יש ממוצע מעל 90 ולפחות הורה אחד שלמד פחות מ-10 שנים',
              LESS_THAN_10_COUNTER:3, 'ל-י');
   WRITELN
 ('סטודנטים יש ממוצע מעל 90 ולפחות הורה אחד שלמד יותר מ-10 שנים',
              MORE_THAN_10_COUNTER:3, 'ל-י')
END.
```

<u>הסבר:</u>

מיון של הסטודנטים לפי מספר הילדים במשפחה. לאחר המיון, 5 הסטודנטים מהמשפחות הקטנות יהיו בתחילת המערך ו-5 סטודנטים מהמשפחות הגדולות ביותר יהיו בסופו.

שים לב שיש לפחות 10 סטודנטים (נתון בשאלה).

תוכנית 2

```
PROGRAM MARKS (INPUT, OUTPUT);
CONST
   MAX_STUDENTS = 100;
   MAX_COURSE   = 100;
TYPE
   CHR_STRING  = PACKED ARRAY [1..20] OF CHAR;
   ADDRESS_REC = RECORD                                  { כתובת }
                    NUMBER    : 1..MAXINT;               { מספר בית }
                    STREET,                              { שם רחוב }
                    CITY,                                { שם עיר }
                    ZIP_CODE : CHR_STRING                { מיקוד }
                 END;
   MARKS = RECORD                                        { ציון }
              NAME : CHR_STRING;                         { שם סטודנט או קורס }
              MARK : 0..100                              { ציון }
           END;
   COURSE = RECORD                                       { קורס }
               C_NAME         : CHR_STRING;              { שם קורס }
               STUDENT_NUMBER : 1..50;                   { מספר סטודנטים }
               RESULTS        : ARRAY [1..50] OF MARKS { מערך }
            END;                   { ציונים המכיל את שם הסטודנט }
   REPORT = RECORD                                       { דו"ח ציונים }
               NUMBER : 1..MAX_COURSE;                   { מספר קורסים }
               RESULT : ARRAY [1..MAX_COURSE] OF MARKS { מערך }
            END;                   { של ציונים המכיל את שם הקורס }
   STUDENT_REC = RECORD                                  { רשומת סטודנט }
                    NAME         : CHR_STRING;           { שם }
                    ID           : REAL;                 { מספר }
                    ADDRESS      : ADDRESS_REC;          { כתובת }
                    MARKS_REPORT : REPORT                { דו"ח ציונים }
                 END;
```

```
VAR
   STUDENT     : ARRAY [1..MAX_STUDENTS] OF STUDENT_REC;
   COURSE_LIST : ARRAY [1..MAX_COURSE] OF COURSE;
   I, J, LAST_STUDENT, LAST_COURSE  : INTEGER;
                                                          { קריאת }
PROCEDURE READ_STUDENT (INDEX : INTEGER);        { נתוני הסטודנטים }
   BEGIN
      WITH  STUDENT [INDEX] , ADDRESS  DO
       BEGIN
         WRITELN;
         WRITELN ('שם סטודנט':20, 'מספר סטודנט':20);
         READLN (NAME, ID);
         WRITELN ('עיר ':20, 'שם רחוב':20, 'מספר':20,
                  'מיקוד':7);
         READLN (CITY, STREET, NUMBER, ZIP_CODE);
         MARKS_REPORT..NUMBER := 0          { איפוס רשימת ציונים }
       END   { with }
   END;   { read_student }
                                                          { הדפסת }
PROCEDURE PRINT_STUDENT (INDEX : INTEGER);          { רשומת סטודנט }
   VAR
      J : INTEGER;
   BEGIN
      WITH  STUDENT [INDEX] , ADDRESS  DO
       BEGIN
         WRITELN;
         WRITELN (NAME:60, 'שם: ':19);
         WRITELN (ID:60:0, 'מספר סטודנט: ':19);
         WRITELN (NUMBER:39, STREET:21, 'כתובת:':19);
         WRITELN (ZIP_CODE:40, CITY:20);
         WRITELN;
         WRITELN ('שם הקורס          ציון ':75);
         WITH  MARKS_REPORT  DO
            FOR  J := 1  TO  NUMBER  DO
               WITH  RESULT [J]  DO
                  WRITELN (MARK:55, NAME:24)
       END   { with }
   END;    { print_student }

PROCEDURE READ_COURSE (INDEX : INTEGER);      { קריאת רשומת קורס }
   VAR
      J : INTEGER;
   BEGIN
      WITH  COURSE_LIST [INDEX]  DO
       BEGIN
         WRITELN;
         WRITELN ('שם הקורס':20, 'מספר הסטודנטים':20);
         READLN (C_NAME, STUDENT_NUMBER);
         WRITELN ('שם סטודנט':20, 'ציון':6);
         FOR  J := 1  TO  STUDENT_NUMBER  DO
            READLN (RESULTS [J].NAME, RESULTS [J].MARK)
```

```
         END    { with }
      END;    { read_course }

   PROCEDURE SORT;                    { מיון הסטודנטים לפי שם - מיון בועות }
      VAR
         STD, J, L : INTEGER;
         X         : STUDENT_REC;
      BEGIN
         J := I;
         WHILE  J > 0  DO
          BEGIN
            L := 0;
            FOR  STD := 1  TO  J - 1  DO
              IF  STUDENT [STD].NAME > STUDENT [STD + 1].NAME  THEN
               BEGIN
                 X := STUDENT [STD];
                 STUDENT [STD] := STUDENT [STD + 1];
                 STUDENT [STD + 1] := X;
                 L := STD
               END;
            J := L
          END    { while }
      END;    { sort }

{ חיפוש בינרי במערך ממויין לפי שם סטודנט. אם נמצא שם, הפונ'
  תחזיר ערך TRUE והפרמטר המשתנה MID יכיל את מספר הרשומה שנמצאה. }

   FUNCTION NAME_FOUND (STUDENT_NAME : CHR_STRING;
                        LAST         : INTEGER;
                    VAR MID          : INTEGER)     : BOOLEAN;
     VAR
        FIRST : INTEGER;
     BEGIN
       FIRST := 1;
       REPEAT
          MID := (FIRST + LAST) DIV 2;
          IF  STUDENT_NAME < STUDENT [MID].NAME THEN
             LAST := MID - 1
          ELSE
             FIRST := MID + 1
       UNTIL (STUDENT_NAME = STUDENT [MID].NAME) OR
                                               (FIRST > LAST);
       NAME_FOUND := STUDENT_NAME = STUDENT [MID].NAME
     END;    { name_found }
```

```
                                                      { עדכון של ציון }
PROCEDURE UPDATE_MARKS (I : INTEGER);            { קורס ברשומת סטודנט }
   VAR
      J, INDEX : INTEGER;
   BEGIN
      WITH  COURSE_LIST [I]  DO
       BEGIN
         FOR  J := 1  TO  STUDENT_NUMBER  DO
            WITH  RESULTS [J]  DO
               IF NAME_FOUND (NAME, LAST_STUDENT, INDEX) THEN
                  WITH  STUDENT [INDEX].MARKS_REPORT  DO
                   BEGIN
                     NUMBER                := NUMBER + 1;
                     RESULT [NUMBER].NAME := C_NAME;
                     RESULT [NUMBER].MARK := MARK
                   END    { with }
       END   { with course_list }
   END;    { update_marks }

BEGIN   { main }
   WRITELN ('מספר הסטודנטים? ');
   READLN (LAST_STUDENT);
   FOR  I := 1  TO  LAST_STUDENT  DO
      READ_STUDENT (I);
   WRITELN ('מספר הקורסים? ');
   READLN (LAST_COURSE);
   FOR  I := 1  TO  LAST_COURSE  DO
      READ_COURSE (I);
   SORT;
   FOR  I := 1  TO  LAST_COURSE  DO
      UPDATE_MARKS (I);
   FOR  I := 1  TO  LAST_STUDENT  DO
      PRINT_STUDENT (I)
END.
```

<u>שיטת הפתרון:</u>

פותרים את הבעיה בשני שלבים:

א. מיון רשימת הסטודנטים לפי שם.
ב. מעבר על רשימת הקורסים.

כאשר עוברים על הרשימות של כל קורס בוחנים את השמות של הסטודנטים ומאתרים את הרשומה הדרושה, בעזרת הפונקציה NAME_FOUND.
אם הרשומה נמצאה: מוסיפים את הציון ואת שם הקורס לדו"ח הציונים של הסטודנט.
אם הסטודנט לא נמצא, מתעלמים מהציון.

הסבר לפרוצדורה UPDATE_MARKS:

126 מעבר על רשימת הסטודנטים וציוניהם.
128 איתור רשומת הסטודנט.
129 הוספת ציון לדו"ח ציונים.
131 עדכון מספר הקורסים.
132 שם הקורס.
133 ציון.

תוכנית 3

```
PROGRAM BANK (INPUT, OUTPUT);
CONST
   MAX_ACCOUNT  = 100;
   MAX_MOVEMENT = 200;
   LAST_MOVE    = -1;
   LAST_ACCOUNT = 'XXXXXXXXXXXXXXX';
TYPE
   CHR_STRING  = PACKED ARRAY [1..15] OF CHAR;
   ACCOUNT_REC = RECORD                                    { רשומת חשבון }
                    NAME   : CHR_STRING;
                    NUMBER : INTEGER;
                    CREDIT,
                    REMAIN : REAL
                 END;
   MOVE_MONEY  = RECORD                                    { רשומת תנועה }
                    ACCOUNT_NUMBER : INTEGER;
                    MONEY          : REAL
                 END;
VAR
   ACCOUNT  : ARRAY [1..MAX_ACCOUNT] OF ACCOUNT_REC;
   MOVEMENT : ARRAY [1..MAX_MOVEMENT] OF MOVE_MONEY;
   ACCOUNTS, MOVES, I : INTEGER;
                                                  { קריאת נתוני חשבון }
FUNCTION READ_ACCOUNT (N : INTEGER) : BOOLEAN;
   BEGIN
      WITH  ACCOUNT [N]  DO
       BEGIN
         READLN (NAME, NUMBER, CREDIT, REMAIN);
         IF  NAME = LAST_ACCOUNT  THEN
            READ_ACCOUNT  := FALSE
         ELSE
            READ_ACCOUNT  := TRUE
       END
   END;     { read_account }
```

```
PROCEDURE PRINT_ACCOUNT (N : INTEGER);      { הדפסת נתוני חשבון }
   VAR
      I : INTEGER;
   BEGIN
      WRITELN ('שם לקוח   ':15, 'מס. חשבון':10, 'אשראי ':8,
                                        'יתרה נוכחית':13);
      FOR  I := 1  TO  N  DO
        WITH  ACCOUNT [I]  DO
           WRITELN (NAME:15, NUMBER:8, CREDIT:10:2, REMAIN:10:2)
   END;    { print_account }
                                                      { קריאת }
FUNCTION READ_MOVE (N : INTEGER) : BOOLEAN;     { נתוני תנועה }
   BEGIN
      WITH  MOVEMENT [N]  DO
       BEGIN
         READLN (ACCOUNT_NUMBER, MONEY);
         READ_MOVE :=  ACCOUNT_NUMBER = LAST_MOVE;
       END
   END;    { read_move }

PROCEDURE SORT_ACCOUNTS;                   { מיון לפי מספר חשבון }
   VAR
      ACC, J, L : INTEGER;
      X : ACCOUNT_REC;
   BEGIN
      J := ACCOUNTS;
      WHILE  J > 0  DO
       BEGIN
         L := 0;
         FOR  ACC := 1  TO  J - 1  DO
           IF ACCOUNT [ACC].NUMBER > ACCOUNT [ACC+1].NUMBER THEN
            BEGIN
              X := ACCOUNT [ACC];
              ACCOUNT [ACC] := ACCOUNT [ACC + 1];
              ACCOUNT [ACC + 1] := X;
              L := ACC
            END;
         J := L
       END   { while j }
   END;    { sort_account }
                                                   { עדכון היתרה }
PROCEDURE UPDATE_MOVE (VAR MOV : MOVE_MONEY);       { לפי התנועה }
   VAR
      INDEX : INTEGER;
```

```
                  { חיפוש בינרי של מספר חשבון,                  }
                  { הפרמטר המשתנה MID מחזיר את מיקום רשומת החשבון }
   FUNCTION ACCOUNT_FOUND (NUMBER : INTEGER;
                   VAR MID    : INTEGER) : BOOLEAN;
      VAR
         FIRST, LAST : INTEGER;
      BEGIN    { account_found }
         FIRST := 1;
         LAST := ACCOUNTS;
         REPEAT
            MID := (FIRST + LAST) DIV 2;
            IF  NUMBER < ACCOUNT [MID].NUMBER THEN
               LAST := MID - 1
            ELSE
               FIRST := MID + 1
         UNTIL
           (NUMBER = ACCOUNT [MID].NUMBER) OR (FIRST > LAST);
         IF  NUMBER = ACCOUNT [MID].NUMBER  THEN
            ACCOUNT_FOUND := TRUE
         ELSE
            ACCOUNT_FOUND := FALSE
      END;    { account_found }

   BEGIN  { update_move }
      WITH  MOV  DO
         IF  ACCOUNT_FOUND (ACCOUNT_NUMBER, INDEX)  THEN
                                        { אם נמצאה רשומת חשבון }
            WITH  ACCOUNT [INDEX]  DO   { יש לעדכן את היתרה.    }
               REMAIN := REMAIN + MONEY
   END; { update_move }

BEGIN { main }
   ACCOUNTS := 0;
   MOVES := 0;
   WRITELN ('שם לקוח':15, 'מס. חשבון':10, 'אשראי':8,
                                    'יתרה נוכחית':13);
   WHILE  READ_ACCOUNT (ACCOUNTS + 1)  DO
      ACCOUNTS := ACCOUNTS + 1;
   WRITELN ('תנועה   מס. חשבון');
   WHILE  READ_MOVE (MOVES + 1)  DO
      MOVES := MOVES + 1;
   SORT_ACCOUNTS;
   FOR  I := 1  TO  MOVES  DO
      UPDATE_MOVE (MOVEMENT [I]);
   PRINT_ACCOUNT (ACCOUNTS)
END.
```

הסבר:

מספר החשבון הוא מספר שלם. למספרי חשבון הגדולים מ-MAXINT יש להשתמש בסוג REAL או בסוג "PACKED ARRAY OF CHAR".

תוכנית 4

```
PROGRAM LIBRARY (INPUT, OUTPUT);
CONST
   MAX_BOOKS   = 20;
   MAX_AUTHORS = 5;
TYPE
   NAME_TYPE  = PACKED ARRAY [1..15] OF CHAR;
   HEAD_TYPE  = PACKED ARRAY [1..70] OF CHAR;
   BOOK_REC   = RECORD                                  { רשומת ספר }
                  HEADER          : HEAD_TYPE;          { שם הספר }
                  AUTHORS_NUMBER  : 1..MAX_AUTHORS;     { מס' מחברים }
                  AUTHOR_NAME                           { מערך שמות המחברים }
                       : ARRAY [1..MAX_AUTHORS] OF NAME_TYPE;
                  PUBLISHER       : NAME_TYPE;          { שם המו"ל }
                  YEAR,                                 { שנת הפרסום }
                  SERIAL          : INTEGER             { מספר סידורי }
                END;
VAR
   BOOK         : ARRAY [1..MAX_BOOKS] OF BOOK_REC;
   BOOKS_NUMBER : INTEGER;
                                                        { קריאת נתוני ספר }
FUNCTION READ_BOOK (VAR BOOK : BOOK_REC) : BOOLEAN;
   VAR
      I : INTEGER;
   BEGIN
      WITH  BOOK  DO
       BEGIN
         WRITELN ('כותרת הספר:':70);
         READLN (HEADER);
         IF  HEADER [1] <> '.'  THEN
          BEGIN
            WRITELN ('מספר המחברים:':70);
            READLN (AUTHORS_NUMBER);
            WRITELN ('שמות המחברים:':70);
            FOR  I := 1  TO  AUTHORS_NUMBER  DO
               READLN (AUTHOR_NAME [I]);
            WRITELN ('שם המוציא לאור:':70);
            READLN (PUBLISHER);
            WRITELN ('שנת ההוצאה לאור:':70);
            READLN (YEAR);
            WRITELN ('מספר סידורי:':70);
            READLN (SERIAL);
            READ_BOOK := TRUE
          END
         ELSE
            READ_BOOK := FALSE
       END    { with book }
   END;    { read_book }
```

```
                                                  { הדפסת נתוני }
PROCEDURE PRINT_BOOKS (BOOKS_NUMBER : INTEGER);   { כל הספרים.  }
   VAR
      I, J : INTEGER;
   BEGIN
      FOR  J := 1  TO  BOOKS_NUMBER  DO
         WITH  BOOK [J]  DO
          BEGIN
            WRITELN (HEADER:70, 'כותרת:':9);
            WRITELN ('מחברים:':70);
            FOR  I := 1  TO  AUTHORS_NUMBER  DO
               WRITELN (AUTHOR_NAME [I]:70);
            WRITELN (PUBLISHER:70, 'מו"ל:':9);
            WRITELN (YEAR:67, 'שנת הוצאה:':12);
            WRITELN (SERIAL:67, 'מס. סידורי:':12)
          END    { with book }
   END;     { print_books }
                                   { מיון לפי שנת ההוצאה לאור }
PROCEDURE SORT_BOOKS (BOOKS_NUMBER : INTEGER);
   VAR
      BOOK_N, L : INTEGER;
      X         : BOOK_REC;
   BEGIN
      WHILE  BOOKS_NUMBER > 0  DO
       BEGIN
         L := 0;
         FOR  BOOK_N := 1  TO  BOOKS_NUMBER - 1  DO
            IF  BOOK [BOOK_N].YEAR > BOOK [BOOK_N+1].YEAR  THEN
             BEGIN
               X                  := BOOK [BOOK_N];
               BOOK [BOOK_N]      := BOOK [BOOK_N + 1];
               BOOK [BOOK_N + 1]  := X;
               L := BOOK_N
             END;
         BOOKS_NUMBER := L
       END  { while }
   END;     { sort_books }

BEGIN { main }
   BOOKS_NUMBER := 0;
   WHILE  READ_BOOK (BOOK [BOOKS_NUMBER + 1])  DO
      BOOKS_NUMBER := BOOKS_NUMBER + 1;
   SORT_BOOKS (BOOKS_NUMBER);
   PRINT_BOOKS (BOOKS_NUMBER)
END.
```

<u>הסבר:</u>

הפונקציה READ_BOOK קוראת נתונים של ספר. במהלך התוכנית היא מקבלת בכל קריאה פרמטר משתנה, שהוא איבר במערך BOOK, ומכניסה לתוכו את הנתונים שנקראו. כאשר מגיעים לסוף הנתונים, הפרוצדורה SORT_BOOKS שמקבלת כפרמטר

את מספר הספרים, ממיינת אותם לפי שנת ההוצאה (מיון בועות). לאחר המיון, הפרוצדורה PRINT_BOOKS מקבלת כפרמטר את מספר הספרים ומדפיסה אותם.

תוכנית 5

```
PROGRAM TEXT_LINES (INPUT, OUTPUT);
CONST
   MAX_LINE_LENGTH = 80;
   MAX_TEXT_LENGTH = 8000;
TYPE
   CHR_NUMBER = 1..MAX_TEXT_LENGTH;
   STR        = PACKED ARRAY [1..MAX_TEXT_LENGTH] OF CHAR;
   CHR_STR    = RECORD
                   LENGTH : CHR_NUMBER;            { אורך התמליל }
                   TEXT   : STR                    {      התמליל }
                END;
   TEXT_LINE  = PACKED ARRAY [1..MAX_LINE_LENGTH] OF CHAR;
VAR
   LINE : TEXT_LINE;
   PAGE : CHR_STR;
   INDEX, LINE_COUNTER, PRESENT_POSITION : CHR_NUMBER;

PROCEDURE READ_TEXT;           { 16 קריאת התמליל: ראה פרק 6 תוכנית }
   BEGIN
      PRESENT_POSITION := 0;
      REPEAT
         LINE_COUNTER := MAX_LINE_LENGTH;
         READLN (LINE);
         WHILE (LINE_COUNTER > 0) AND (LINE [LINE_COUNTER] = ' ')
            DO LINE_COUNTER := LINE_COUNTER - 1;
         FOR INDEX := 1  TO  LINE_COUNTER  DO
            PAGE.TEXT [PRESENT_POSITION + INDEX] := LINE [INDEX];
         PRESENT_POSITION := PRESENT_POSITION + LINE_COUNTER + 1;
         PAGE.TEXT [PRESENT_POSITION] := ' '
      UNTIL  LINE[1] = '.';
      PAGE.LENGTH := PRESENT_POSITION - 2;
      WRITELN
   END;   { read_text }

PROCEDURE PRINT_SENTENCES;
   CONST
      QUEST = '?';
      EXCL  = '!';
      POINT = '.';
   VAR
      I : CHR_NUMBER;
```

```
                                        { השוואה בין המקום FROM בתמליל לתו FIN, }
    FUNCTION EQUAL (FROM : CHR_NUMBER;  { ובדיקה שאחריו אין רווח }
                    FIN  : CHAR       ) : BOOLEAN;
    BEGIN
       WITH  PAGE  DO
         EQUAL := (TEXT [FROM] = FIN) AND (TEXT [FROM + 1] = ' ');
    END;     { equal }
                                                              { זיהוי סוף }
    FUNCTION END_SENTENCE (FROM : CHR_NUMBER) : BOOLEAN;  { משפט }
       BEGIN
        END_ SENTECNE := EQUAL (FROM, QUEST) OR
                         EQUAL (FROM, EXCL)  OR
                         EQUAL (FROM, POINT)
       END;    { end_sentence }

    BEGIN { print_sentences }
       I := 1;
       WITH  PAGE  DO
          WHILE  I <= LENGTH  DO
           BEGIN
             WRITE (TEXT [I]);                         { הדפסת תו }
             IF  END_SENTENCE (I)  THEN                { סיום משפט }
              BEGIN
                WRITELN;                               { שורה חדשה }
                I := I + 1                { דילוג על רווח סוף משפט }
              END;
             I := I + 1                                { התו הבא }
          END  { while i <= }
    END;     { print_sentences }

BEGIN { main }
    READ_TEXT;
    PRINT_SENTENCES
END.
```

הסבר:

* הפרוצדורה READ_TEXT קוראת את התמליל לתוך מערך PAGE (ראה תוכנית 16 בפרק 6).
* הפרוצדורה PRINT_SENTENCE מדפיסה את התמליל לפי משפטים - כל משפט בשורה.
* סוף משפט מזוהה על ידי הפונקציה END_SENTENCE, אשר נעזרת בפונקציה EQUAL.
* פונקציה EQUAL משווה את תווי התמליל לתווים המסמנים סוף משפט.

פרק 11
קבצים

<u>הערה:</u>

בתוכניות הבאות מופיעה הפקודה:

ASSIGN (משתנה-קובץ, מחרוזת);

פקודה זו משייכת שם קובץ פיסי למשתנה מסוג קובץ.
פקודת ASSIGN אינה שייכת לפסקל הסטנדרטי ולכן יתכן, שבמהדרים שונים תהיינה פקודות שונות למטרה זו.

תוכניות

תוכניות 1 ו-2

```
PROGRAM FILE_STUDENTS (INPUT, OUTPUT, STUDENTS);
TYPE                                                    { רשומת סטודנט }
   STUDENT_REC = RECORD
                  NAME    : PACKED ARRAY [1..15] OF CHAR;    { שם }
                  ID,                                         { מספר זהות }
                  AVERAGE : REAL;                             { ממוצע }
                  SEX     : (Z, N);                           { מין }
                  HOURS   : INTEGER                           { מספר שעות }
                 END;
   STUDENTS = FILE OF STUDENT_REC;        { קובץ המורכב מרשומות סטודנט }
VAR
   STD     : STUDENTS;
   STUDENT : STUDENT_REC;
BEGIN
   ASSIGN (STD, 'STUDENT.DAT');
   RESET (STD);
```

```
   WHILE  NOT EOF (STD)  DO
    BEGIN
      READ (STD, STUDENT);
      WITH  STUDENT  DO
         IF  ((SEX = N) AND (AVERAGE > 75)) OR
             ((SEX = Z) AND (AVERAGE < 75)) THEN
            WRITELN (NAME)
    END;
   CLOSE (STD)
END.
```

תוכנית 3

```
PROGRAM SHLIFOT (OUTPUT, EMPLOYEE);
TYPE

   EMPLOY_REC =                                             { רשומת עובד }
      RECORD
         SALARY     : REAL;                                 {   משכורת }
         STATUS     : (RAVAK, NASUI, GARUSH, ALMAN);        { מצב משפ' }
         SEX        : (N, Z);                               {      מין }
         EDUCATION  : (YESODI, TICHON, RISHON, SHENI);      {    השכלה }
         PROFESSION : (MECHANIC, ELECTRONIC, COMPUTER);     {     מקצוע }
         NAME       : PACKED ARRAY [1..20] OF CHAR;         {        שם }
         NUMBER     : INTEGER                               {     מספר }
      END;

   EMPLOY_FILE = FILE OF EMPLOY_REC;                   { קובץ רשומות עובד }

VAR
   EMPLOYEE : EMPLOY_FILE;
   EMP      : EMPLOY_REC;

FUNCTION READ_EMPLOYEE : BOOLEAN;
   BEGIN
         { קריאת נתונים של רשומת עובד למשתנה EMP לפי דוגמאות בפרק 10
                   למשל, לפי הפונקציה READ_BOOK בתוכנית 4 שבפרק 10. }
   END;

    { פרוצדורה CREATE_FILE יוצרת קובץ, קוראת נתונים של רשומות עובדים
       וכותבת אותם בקובץ. התהליך נמשך כל עוד הפונקציה READ_EMPLOYEE
                           מחזירה ערך TRUE. כלומר - יש עוד קלט. }
PROCEDURE  CREATE_FILE;
   BEGIN
      REWRITE (EMPLOYEE);
      WHILE  READ_EMPLOYEE  DO
         WRITE (EMPLOYEE, EMP);
      CLOSE (EMPLOYEE)
   END;     { create_file }
```

```
PROCEDURE PRINT_MALE;          { פרוצדורה לשליפת שמות העובדים ממין זכר }
   BEGIN
      RESET (EMPLOYEE);
      WHILE  NOT EOF (EMPLOYEE)  DO
       BEGIN
         READ (EMPLOYEE, EMP);
         WITH  EMP  DO
            IF  SEX = Z  THEN
               WRITELN (NAME)
       END;
      CLOSE (EMPLOYEE)
   END;       { print_male }

 FUNCTION CALCULATE_AVERAGE : REAL;  { פרוצדורה לחישוב ממוצע המשכורות }
    VAR
       SUM     : REAL;
       COUNTER : INTEGER;
    BEGIN
       COUNTER := 0;
       SUM     := 0;
       RESET (EMPLOYEE);
       WHILE  NOT EOF (EMPLOYEE)  DO
        BEGIN
          READ (EMPLOYEE, EMP);
          WITH  EMP  DO
             SUM := SUM + SALARY;
          COUNTER := COUNTER + 1
        END;
       CLOSE (EMPLOYEE);
       IF  COUNTER > 0  THEN
          CALCULATE_AVERAGE := SUM / COUNTER
       ELSE                                                  { במקרה זה SUM = 0 }
          CALCULATE_AVERAGE := SUM
    END;      { calculate_average }

        { פרוצדורה להדפסה של שמות העובדים שמשכורתם גבוהה מהפרמטר LIMIT }

   PROCEDURE PRINT_MORE_THAN (LIMIT : REAL);
      BEGIN
         RESET (EMPLOYEE);
         WHILE  NOT EOF (EMPLOYEE)  DO
          BEGIN
            READ (EMPLOYEE, EMP);
            WITH  EMP  DO
               IF  SALARY > LIMIT  THEN
                  WRITELN (NAME)
          END;
         CLOSE (EMPLOYEE)
      END;     { print_more_than }
```

```
{ פרוצדורה להדפסת שמות העובדים שמשכורתם גבוהה מ-900 }

PROCEDURE PRINT_900;
   BEGIN
      PRINT_MORE_THAN (900)
   END;

{ פרוצדורה להדפסת שמות העובדים
  שמשכורתם גבוהה ממוצע המשכורות. }

PROCEDURE PRINT_AVERAGE;
   BEGIN
      PRINT_MORE_THAN (CALCULATE_AVERAGE)
   END;

{ הפרוצדורה להדפסה של שמות העובדים שהם מתכנתים בעלי
  תואר ראשון, או עובדים שמספר הזיהוי שלהם גדול מ-1500. }

PROCEDURE PRINT_PART_E;
   BEGIN
      RESET (EMPLOYEE);
      WHILE  NOT EOF (EMPLOYEE)  DO
       BEGIN
         READ (EMPLOYEE, EMP);
         WITH  EMP  DO
            IF  ((PROFESSION = COMPUTER) AND (EDUCATION = RISHON))
             OR ((STATUS= RAVAK) AND (NUMBER > 1500))  THEN
               WRITELN (NAME)
       END;
      CLOSE (EMPLOYEE)
   END;   { print_part_e }

BEGIN  { main }
   ASSIGN (EMPLOYEE, 'EMP.DAT');

   { קריאה לפרוצדורות לפי הצורך }

END.
```

<u>הסבר:</u>

בתוכנית זו ישנן פרוצדורות אחדות אשר שולפות נתונים מקובץ העובדים.
המבנה של כל אחת מהן הוא:

א. פתיחת הקובץ.
ב. הזזת המצביע לרשומה הראשונה בקובץ.
ג. סריקה עד סוף הקובץ תוך כדי בדיקת התנאי הדרוש. אם התנאי מתקיים - מדפיסים את שם העובד.
ד. סגירת הקובץ.

תוכנית 6

```
PROGRAM DRINK_PRICE (DRINKS, ORDERS, OUTPUT);
CONST
   MAX_DRINK_KIND = 4;
TYPE
   NAME       = PACKED ARRAY [1..12] OF CHAR;

   DRINK_REC = RECORD                              { רשומת משקה }
                  DRINK_NAME : NAME;               { שם המשקה }
                  PRICE_1,                         { מחיר ארגז אחד }
                  PRICE_6,                         { מחיר שישה ארגזים }
                  PRICE_12,                        { מחיר שנים עשר ארגזים }
                  PRICE_24   : INTEGER             { מחיר עשרים וארבע ארגזים }
               END;

   ONE_ORDER = RECORD                              { רשומת הזמנה בודדת }
                  DRINK_NAME : NAME;               { שם המשקה }
                  BOX_NUMBER : INTEGER             { מספר ארגזים }
               END;

   ORDER_REC = RECORD                              { רשומת הזמנה }
                  ORDER_NUMBER : INTEGER;          { מספר הזמנה }
                  ORDER_BY     : NAME;             { שם המזמין }
                  ADDRESS      : PACKED ARRAY [1..30] OF CHAR;{ כתובת }
                  CITY         : NAME;             { עיר }
                  ZIP_CODE     : REAL;             { מיקוד }
                                                   { רשימת הזמנות בודדות }
                  ORDER_LIST   : ARRAY [1..MAX_DRINK_KIND] OF ONE_ORDER
               END;

   DRINK_FILE = FILE OF DRINK_REC;                 { קובץ משקאות }
   ORDER_FILE = FILE OF ORDER_REC;                 { קובץ הזמנות }
VAR
   DRINKS : DRINK_FILE;
   ORDERS : ORDER_FILE;
   DRINK  : DRINK_REC;
   ORDER  : ORDER_REC;
   I,
   PRICE,
   TOTAL  : INTEGER;
```

```
                                    { הפונקציה סורקת את קובץ המשקאות
                                    ומחזירה TRUE אם נמצאה רשומה של משקה N. }
FUNCTION FIND_RECORD (VAR DRINK : DRINK_REC;    N : NAME) : BOOLEAN;
   BEGIN
      RESET (DRINKS);
      REPEAT
         READ (DRINKS, DRINK)
      UNTIL  (DRINK.DRINK_NAME = N)  OR  EOF (DRINKS);
      FIND_RECORD := DRINK.DRINK_NAME = N
   END;   { find_record }

BEGIN { main }
   ASSIGN (DRINKS, 'DRINK.DAT');
   RESET (DRINKS);
   ASSIGN (ORDERS, 'ORDER.DAT');
   RESET (ORDERS);
   WHILE  NOT EOF (ORDERS)  DO                  { לולאה על כל קובץ ההזמנות }
    BEGIN
      READ (ORDERS, ORDER);                     { קרא הזמנה.        }
      WRITELN;
      WITH  ORDER  DO
       BEGIN                                    { הדפסת פרטי ההזמנה  }
         WRITELN ('ORDER NUMBER: ',ORDER_NUMBER:1,
                  'ORDERED BY: ':30, ORDER_BY:12);
         WRITELN ('ADDRESS: ', ADDRESS:30, CITY:14, ZIP_CODE:8:0);
         WRITELN ('  DRINK NAME         BOXES      PRICE');
         WRITELN ('-----------------------------------');
         I := 1;
         TOTAL := 0;
         WHILE  ORDER_LIST [I].BOX_NUMBER > 0  DO
          BEGIN                        { לולאה עד סוף הרשימה ברשומת ההזמנה }
            WITH  ORDER_LIST [I]  DO
               IF  FIND_RECORD (DRINK, DRINK_NAME)  THEN
                BEGIN                           { נמצאה רשומת המחיר של המשקה }
                  WITH  DRINK  DO               { חישוב המחיר של ההזמנה.   }
                     PRICE := (BOX_NUMBER DIV 24) * PRICE_24 +
                        ((BOX_NUMBER MOD 24)  DIV 12) * PRICE_12 +
                        ((BOX_NUMBER MOD 12)  DIV 6) * PRICE_6 +
                        (BOX_NUMBER MOD 6) * PRICE_1;
                  TOTAL := TOTAL + PRICE;       { סיכום מחירי ההזמנות
                                                  הבודדות המרכיבות את ההזמנה. }
                  WRITELN (DRINK_NAME:15, BOX_NUMBER:8, PRICE:10)
                END     { if find_record }
               ELSE                             { הרשומה לא נמצאה }
                  WRITELN (DRINK_NAME:15, 'NOT FOUND')
            I := I + 1
          END;   { while order_list }
         WRITELN ('TOTAL :':23, TOTAL:10)       { הדפסת שווי ההזמנה }
       END    { with order }
    END    { while not eof }
END.    { main }
```

תוכנית 7

בתוכנית זו מופיעה הפקודה (משתנה-קובץ) ERASE, שתפקידה למחוק קובץ פיסי. הפקודה ERASE אינה שייכת לפסקל הסטנדרטי ולכן יתכן שבמהדרים שונים תופיע פקודה שונה למטרה זו.

הסבר למבנה התוכנית:
תוכנית זו מורכבת ממספר חלקים הכוללים פרוצדורות ופונקציות פשוטות ומקוננות, ולבסוף - גוף התוכנית. נציג להלן באופן סכמטי את חלקיה:

```
program BOOK_INDEX
   procedure COPY_FILE
      begin - end
   procedure MERGE_FILES
      begin - end
   procedure SORT
      procedure GET_RECORD
         begin - end
      begin - end   { sort }
   procedure MERGE_SORT
      begin - end
   procedure BUILD_INDEX_FILE
      begin - end
begin - end  { main }
```

```
PROGRAM BOOK_INDEX (INPUT, OUTPUT);
CONST
   MAX_WORD_LENGTH = 15;                    { אורך מלה מכסימלי }
   MAX_CHAP        = 20;                    { מספר מכסימלי של פרקים }

TYPE
   WORD_TYPE  = PACKED ARRAY [1..MAX_WORD_LENGTH] OF CHAR;

   CHAP_INDEX = RECORD             { רשומה בקובץ אינדקס של פרק בודד }
                   WORD    : WORD_TYPE;                    { מלה }
                   PAGE_NU : INTEGER                       { מספר עמוד }
                END;

   BOOK_INDEX = RECORD             { רשומה בקובץ אינדקס של כל הספר }
                   WORD        : WORD_TYPE;                { מלה }
                   LAST_PAGE   : 1..20;      { מספר מופעי המלה בספר }
                   PAGE_NUMBER : ARRAY [1..20] OF INTEGER    { מספרי
                                       העמודים בהם הופיעה המלה. }
                END;

   FILE_NAME    = WORD_TYPE;               { מחרוזת - שם הקובץ הפיסי }
   CHAPTER_FILE = FILE OF CHAP_INDEX;      { קובץ אינדקס של פרק בודד }
   INDEX_FILE   = FILE OF BOOK_INDEX;      { קובץ אינדקס של כל הספר }
```

```
VAR
   CHAP_FILE_NAME : ARRAY [1..MAX_CHAP] OF FILE_NAME;
                                                { מערך של שמות הקבצים }
   CHAP_NUMBER    : 1..MAX_CHAP;                { מספר הפרקים שבספר }
   JUMP, F_N            : INTEGER;

            { פרוצדורה זו מעתיקה את קובץ FROM_FILE לקובץ TO_FILE }

PROCEDURE COPY_FILE (VAR FROM_FILE,
                         TO_FILE     : CHAPTER_FILE);
  VAR
     REC : CHAP_INDEX;
  BEGIN
     RESET (FROM_FILE);
     REWRITE (TO_FILE);
     WHILE  NOT EOF (FROM_FILE)  DO
      BEGIN
        READ (FROM_FILE, REC);
        WRITE (TO_FILE, REC)
      END
  END;      { copy_file }

           { הפרוצדורה ממזגת שני קבצים ממויינים לקובץ אחד ממויין.
                     הקובץ הממויין מחליף את הקובץ הראשון F1_NAME.
                     הפרוצדורה מקבלת את השמות הפיסיים של הקבצים. }

PROCEDURE MERGE_FILES (F1_NAME,
                       F2_NAME   : FILE_NAME);
  VAR
     FILE_1,
     FILE_2,
     TEMP_FILE   : CHAPTER_FILE;
     REC_1,
     REC_2   : CHAP_INDEX;
     READ_FROM_FIRST_FILE : BOOLEAN;      { משתנה בוליאני המצביע על
                                          הקובץ ממנו תקרא הרשומה:
                                          אם TRUE יש לקרוא רשומה מקובץ 1
                                          ואם FALSE יש לקרוא רשומה מקובץ 2. }
  BEGIN   { merge_files }
     ASSIGN (FILE_1, F1_NAME);
     ASSIGN (FILE_2, F2_NAME);
     RESET (FILE_1);
     RESET (FILE_2);
     IF  NOT (EOF (FILE_1) AND EOF (FILE_2)) THEN
      BEGIN                                 { אם שני הקבצים אינם ריקים }
        ASSIGN (TEMP_FILE, 'TEMP');         { פותחים קובץ זמני }
        REWRITE (TEMP_FILE);
        IF  NOT EOF (FILE_1)  THEN
           READ (FILE_1, REC_1);            { קריאת רשומה ראשונה מקובץ 1 }
        READ_FROM_FIRST_FILE := FALSE;      { הקריאה הבאה תהיה מקובץ 2 }
```

```
WHILE  NOT (EOF (FILE_1) AND READ_FROM_FIRST_FILE OR
            EOF (FILE_2) AND NOT READ_FROM_FIRST_FILE)  DO
                      { הלולאה מתבצעת כל זמן שהקובץ ממנו צריכים
                                    .לקרוא את הרשומה הבאה לא הסתיים }
 BEGIN                                    { קריאת רשומה מהקובץ המתאים }
   IF  READ_FROM_FIRST_FILE  THEN
      READ (FILE_1, REC_1)
   ELSE
      READ (FILE_2, REC_2);
   IF  (REC_1.WORD < REC_2.WORD) OR
       ((REC_1.WORD = REC_2.WORD) AND
       (REC_1.PAGE_NU < REC_2.PAGE_NU))  THEN
    BEGIN  {      הרשומה שנקראה מקובץ 1 קודמת לרשומה שנקראה
           .מקובץ 2, ולכן הרשומה מקובץ 1 תכתב לקובץ הזמני
                                         .הרשומה הבאה תקרא מקובץ 1 }
      WRITE (TEMP_FILE, REC_1);
      READ_FROM_FIRST_FILE  := TRUE
    END
   ELSE
    BEGIN  {      הרשומה שנקראה מקובץ 2 קודמת לרשומה שנקראה }
           {.מקובץ 1, ולכן הרשומה מקובץ 2 תכתב לקובץ הזמני
                                         .הרשומה הבאה תקרא מקובץ 2 }
      WRITE (TEMP_FILE, REC_2);
      READ_FROM_FIRST_FILE  := FALSE
    END
 END;    { while }

    { לשלב זה מגיעים כאשר אחד הקבצים הגיע לסופו. יש להעביר
           .את שארית הקובץ שעדיין לא הסתיים אל הקובץ הזמני }
IF  EOF (FILE_1)  THEN
 BEGIN
   WHILE  NOT EOF (FILE_2)  DO
    BEGIN                       { פקודת WRITE הראשונה שתבוצע תכתוב }
                                { .WHILE-את הרשומה שנשארה מלולאת ה }
      WRITE (TEMP_FILE, REC_2);
      READ (FILE_2, REC_2)
    END;
   WRITE (TEMP_FILE, REC_2)
 END
ELSE
 BEGIN
   WHILE  NOT EOF (FILE_1)  DO
    BEGIN
      WRITE (TEMP_FILE, REC_1);
      READ (FILE_1, REC_1)
    END;
   WRITE (TEMP_FILE, REC_1)
 END;
```

```
      COPY_FILE (TEMP_FILE, FILE_1);      { העתקה של הקובץ הזמני }
                                          { לקובץ 1. }
      CLOSE (TEMP_FILE);
      ERASE (TEMP_FILE)                   { ראה הערה בתחילת התוכנית }
    END;   { if }
   CLOSE (FILE_1);
   CLOSE (FILE_2)
 END;    { merge_files }

     { הפרוצדורה SORT מקבלת שני קבצים (F1, F2) שממויינים בבלוקים
     בגודל K, ומחזירה שני קבצים (G1, G2) הממויינים בבלוקים של 2K.
     זה נעשה ע"י מיזוג על שני בלוקים בגודל K מ-F1 ו-F2 וכתיבתם
     לקבצים G1 ו-G2 לסירוגין. }

PROCEDURE SORT (K    : INTEGER;
                VAR F1, F2 , G1, G2  : CHAPTER_FILE);
  VAR
    OUT_SWITCH : BOOLEAN;        { משתנה הקובע לאיזה קובץ תכתב הרשומה }
    WINNER     : 1..2;           { מצביע על הרשומה הקודמת לפי הסדר }
    USED       : ARRAY [1..2] OF INTEGER;     { ספירת מספר הרשומות
                                                שנקראו מכל קובץ. }
    FIN        : ARRAY [1..2] OF BOOLEAN;     { מציין שנקראו כל
                                                הרשומות בבלוק. }
    CURRENT    : ARRAY [1..2] OF CHAP_INDEX;  { מערך שמחסין את שתי
                                                הרשומות שבזיכרון. }

                                  { הפרוצדורה מקבלת מספר קובץ
                                  וקוראת ממנו רשומה אחת. }

  PROCEDURE GET_RECORD (FILE_NUMBER : INTEGER);   { פרוצדורה מקוננת }
  BEGIN
    IF  (USED [FILE_NUMBER] = K) OR ((FILE_NUMBER = 1) AND EOF (F1))
     OR ((FILE_NUMBER = 2) AND EOF (F2))  THEN
      FIN [FILE_NUMBER] := TRUE        { מסמן סוף בלוק או סוף קובץ }
    ELSE
     BEGIN
      USED [FILE_NUMBER] := USED [FILE_NUMBER] + 1;        { ספירת
                          הרשומות שנקראו מקובץ FILE_NUMBER. }
      IF  FILE_NUMBER = 1  THEN  { קריאת רשומה מקובץ FILE_NUMBER }
         READ (F1, CURRENT [FILE_NUMBER])
      ELSE
         READ (F2, CURRENT [FILE_NUMBER])
     END
  END;  { get_record }
```

```
BEGIN   { sort }
   OUT_SWITCH := TRUE;                                    { הפלט לקובץ G1 }
   RESET (F1);                                            { איתחול הקבצים }
   RESET (F2);
   REWRITE (G1);
   REWRITE (G2);
   WHILE  NOT EOF (F1) OR NOT EOF (F2)  DO
    BEGIN                        { כל זמן ששני הקבצים לא הגיעו לסופם }
     USED [1] := 0;
     USED [2] := 0;
     FIN [1]  := FALSE;
     FIN [2]  := FALSE;
     GET_RECORD (1);                                  { קריאת רשומה מקובץ 1 }
     GET_RECORD (2);                                  { קריאת רשומה מקובץ 2 }
     WHILE  NOT FIN [1] OR NOT FIN [2]  DO
      BEGIN            { אם עדיין לא הסתיימה קריאת בלוק בשני הקבצים }
        IF  FIN [1]  THEN                             { אם נגמר בלוק בקובץ 1 }
           WINNER := 2                                { אז קובץ 2 ממשיך. }
        ELSE                                          { לא נגמר בלוק בקובץ 1 }
           IF  FIN [2]  THEN                          { אם נגמר בלוק בקובץ 2 }
              WINNER := 1                             { אז קובץ 1 ממשיך. }
           ELSE                         { לא נגמר בלוק בקובץ 1 ובקובץ 2 }
              IF (CURRENT [1].WORD < CURRENT [2].WORD) OR
                 ((CURRENT [1].WORD = CURRENT [2].WORD) AND
                 (CURRENT [1].PAGE_NU < CURRENT [2].PAGE_NU)) THEN
                                                      { השוואה בין הרשומות. }
                 WINNER := 1                          { רשומה מקובץ 1 קודמת }
              ELSE
                 WINNER := 2;                         { רשומה מקובץ 2 קודמת }
        IF  OUT_SWITCH  THEN        { כתיבה לקבצים G1 ו-G2 לסירוגין }
           WRITE (G1, CURRENT [WINNER])
        ELSE
           WRITE (G2, CURRENT [WINNER]);
        GET_RECORD (WINNER)                           { קריאת רשומה מהקובץ של }
                                                      { הרשומה האחרונה שנכתבה }
      END;    { while not fin }
     OUT_SWITCH := NOT OUT_SWITCH                     { מיזוג שני הבלוקים }
                                                      { הבאים יעשה לקובץ השני. }
    END   { while not eof }
END;  { sort }
```

```
                        { מיון אחד או שני קבצים לא ממויינים
                        לקובץ אחד ממויין שיחליף את F1_NAME }

PROCEDURE MERGE_SORT (F1_NAME,
                      F2_NAME  : FILE_NAME;
                      FILES    : INTEGER);          { מספר הקבצים למיון }
  VAR
    F1, F2 , G1, G2  : CHAPTER_FILE;
    I   : INTEGER;
  BEGIN
    ASSIGN (F1, F1_NAME);
    IF  FILES = 2  THEN
      ASSIGN (F2, F2_NAME)                        { אם יש שני קבצים למיון }
    ELSE
     BEGIN                                        { יש רק קובץ אחד למיון }
      ASSIGN (F2, 'TEMP1');                       { פתיחת קובץ זמני }
      REWRITE (F2)
     END;
    ASSIGN (G1, 'TEMP2');
    ASSIGN (G2, 'TEMP3');
    I := 1;                                       { קביעת גודל הבלוק }
    REPEAT
      { מיון F1 ו-F2 ל-G1 ו-G2. אם יש רק קובץ אחד הפרוצדורה SORT
        תגרום לפיצול הקובץ הבודד לשני קבצים והתהליך ממשיך בדומה
        למקרה של שני קבצים. אורך הבלוק גדל כפול שניים, (פרט למעבר
        הראשון אם היה רק קובץ אחד, אז גודל הבלוק ישאר 1). מיון
        נוסף והפעם G1 ו-G2 ל-F1 ו-F2 ושוב הגדלת אורך הבלוק, פי 2. }
      SORT (I, F1, F2, G1, G2);
      I := I * FILES;
      FILES := 2;
      SORT (I, G1, G2, F1, F2);
      I := I * 2;
      RESET (F2)
    UNTIL  EOF (F2);
     { הלולאה מתבצעת עד שקובץ 2 מתרוקן. כלומר, הבלוק גדול או שווה
       למספר כל הרשומות, ולכן כל הרשומות ממויינות בקובץ הראשון. }
    CLOSE (F1);
    CLOSE (F2);
    CLOSE (G1);
    CLOSE (G2)
  END;    { merge_sort }
```

```
{ פרוצדורה זו בונה את קובץ האינדקס של כל הספר מהקובץ שנבנה קודם.
  אלו הן רשומות מסוג אינדקס של כל הפרקים אשר ממויינות לפי א"ב
  ומיון מישני לפי מספרי עמודים. }

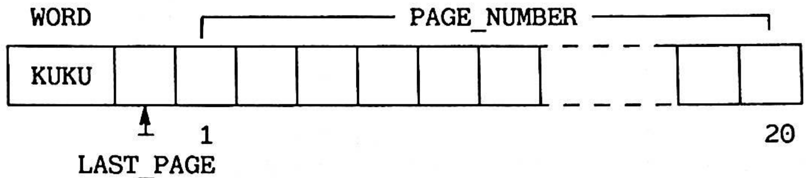
PROCEDURE BUILD_INDEX_FILE;
   VAR
      F            : CHAPTER_FILE;
      C            : CHAP_INDEX;
      BOOK_NDX     : INDEX_FILE;
      B            : BOOK_INDEX;
      TEMP_WORD    : WORD_TYPE;
      BUFFER_FULL  : BOOLEAN;            { מורה על המצאות רשומה בזיכרון }
      I            : INTEGER;
   BEGIN    { build_index_file }
      ASSIGN (F, CHAP_FILE_NAME [1]);
      RESET (F);
      ASSIGN (BOOK_NDX, 'BOOK.NDX');
      REWRITE (BOOK_NDX);
      BUFFER_FULL := FALSE;
      IF  NOT EOF (F)  THEN
       BEGIN
         READ (F, C);                            { קריאת רשומה לזיכרון }
         BUFFER_FULL := TRUE
       END;
      WHILE NOT EOF (F) OR BUFFER_FULL DO  { לולאה כל זמן שלא הסתיים }
       BEGIN                 { הקובץ, או שעדיין ישנה רשומה בזיכרון. }
         TEMP_WORD := C.WORD;                   { מלת האינדקס הנוכחית }
         WITH  B  DO
          BEGIN
            WORD := C.WORD;
            PAGE_NUMBER [1] := C.PAGE_NU;
            LAST_PAGE := 1
          END;
         BUFFER_FULL := FALSE;
         WHILE  NOT EOF (F) AND (TEMP_WORD = C.WORD)  DO
          BEGIN { לולאה כל זמן שלא סוף הקובץ ומופיעה אותה מלה ברשומה }
            READ (F, C);
            BUFFER_FULL := TRUE;
            IF  TEMP_WORD = C.WORD  THEN
             BEGIN                { אם ברשומה הבאה מופיעה אותה מלה }
               BUFFER_FULL := FALSE;
               WITH  B  DO
                BEGIN
                  LAST_PAGE := LAST_PAGE + 1;   { ספירת מספר המופעים }
                  PAGE_NUMBER [LAST_PAGE] := C.PAGE_NU        { הוספת }
                                                      { מספר העמוד. }

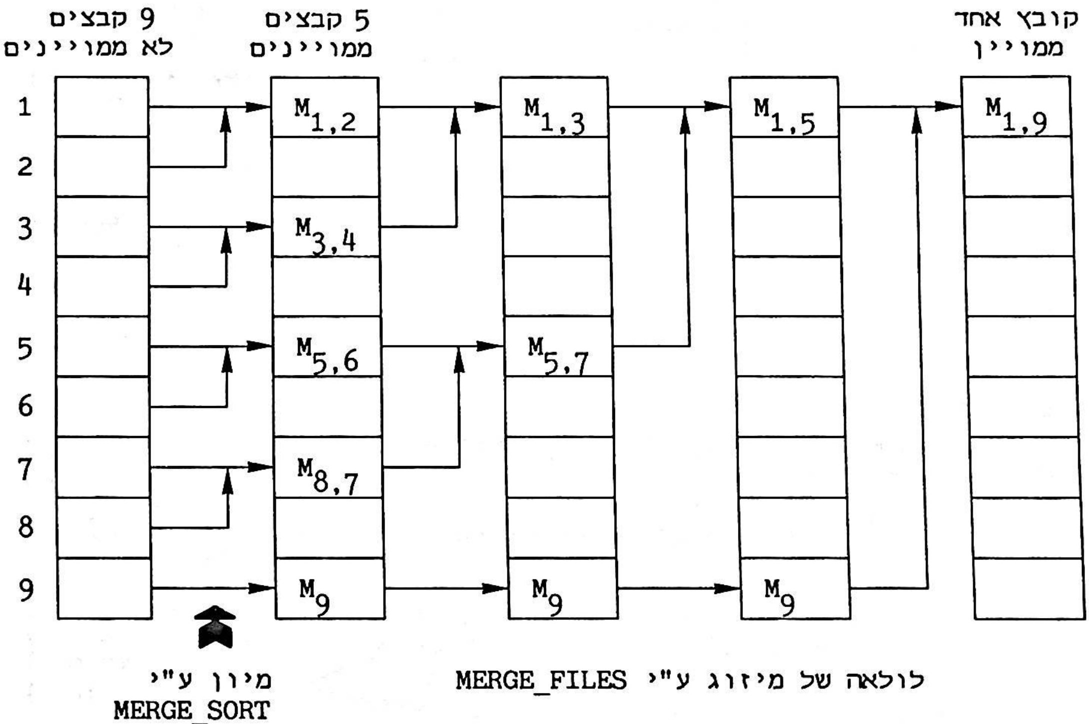
                END
             END   { if temp }
          END;   { while not eof }
         WRITE (BOOK_NDX, B)
       END;   { while not eof }
```

```
      CLOSE (F);
      RESET (BOOK_NDX);
      WRITELN;
      WHILE  NOT EOF (BOOK_NDX)  DO        { הדפסת קובץ אינדקס של הספר }
       BEGIN
         READ (BOOK_NDX, B);
         WITH  B  DO
          BEGIN
            WRITE (WORD);
            FOR I:= 1  TO  LAST_PAGE - 1 DO
               WRITE (PAGE_NUMBER [I]:1, ',');
            WRITELN (PAGE_NUMBER [LAST_PAGE]:1)
          END
       END;    { while not eof }
      CLOSE (BOOK_NDX)
   END;     { build_index_file }

BEGIN  { main }                                        { גוף התוכנית }
   WRITE ('ENTER CHAPTERS NUMBER: ');
   READLN (CHAP_NUMBER);                               { מספר הפרקים }
   WRITELN ('ENTER FILE NAMES OF ALL CHAPTERS');
   FOR  F_N := 1  TO  CHAP_NUMBER  DO
      READLN (CHAP_FILE_NAME [F_N]);        { קריאת שמות הקבצים שבהם }
                                            { נמצא האינדקס של הפרקים. }
   FOR  F_N := 1  TO  CHAP_NUMBER DIV 2  DO  { מיון בזוגות של הקבצים }
      MERGE_SORT (CHAP_FILE_NAME [2 * F_N - 1],
                      CHAP_FILE_NAME [2 * F_N], 2);
   IF  ODD (CHAP_NUMBER)  THEN     { אם מספר הקבצים למיון הוא אי-זוגי }
         MERGE_SORT (CHAP_FILE_NAME [CHAP_NUMBER],      { מיון הקובץ }
                      CHAP_FILE_NAME [CHAP_NUMBER], 1); { האחרון לבד. }
   JUMP := 2;
   WHILE  JUMP < CHAP_NUMBER  DO
    BEGIN                   { מיזוג זוגות קבצים עד שמגיעים לקובץ אחד }
      F_N := 1;
      WHILE  F_N + JUMP <= CHAP_NUMBER  DO
       BEGIN
         MERGE_FILES (CHAP_FILE_NAME [F_N],
                       CHAP_FILE_NAME [F_N + JUMP]);
         F_N := F_N + 2 * JUMP
       END;
      JUMP := 2 * JUMP
   END;    { while jump }
  BUILD_INDEX_FILE                         { בניית האינדקס של הספר }
END.    { main }
```

מבנה רשומה מסוג BOOK_INDEX:





הסבר מפורט על מיון קבצים נמצא בספר "פסקל" (פרק 11.4).

הגירסה המוצגת כאן שונה, מהגירסה המופיעה בספר בכך, שפיצול הקובץ לבלוקים בגודל K נעשה תוך כדי הכתיבה. שוני נוסף הוא, שהמיון יכול להתבצע על קובץ אחד או שני קבצים (שניהם לא ממויינים). הבחירה נעשית ע"י המשתנה FILES שמקבל ערך 1 או 2.

ראה גם תוכנית 4 שבפרק 13, אשר דנה בנושא המיון.

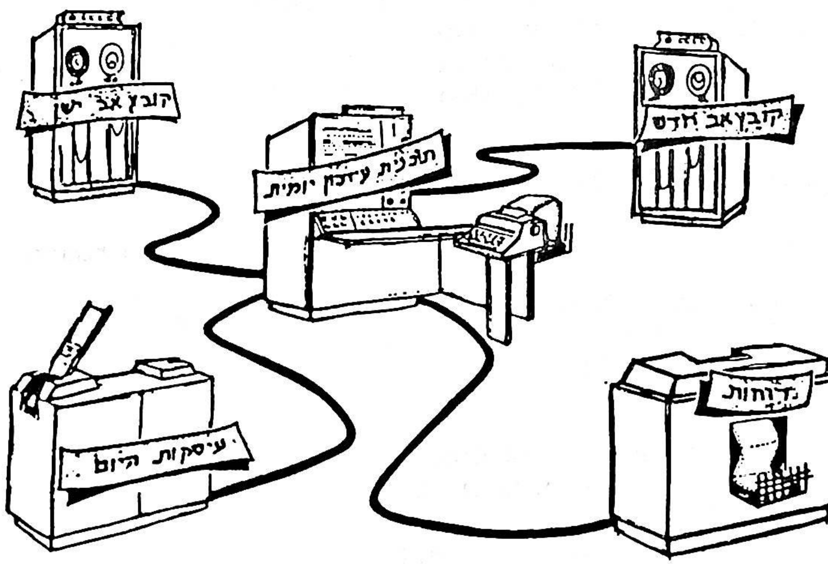
פרק 12
עדכון קבצים

שאלות

שאלה 2

```
TYPE
   CHR_STRING = PACKED ARRAY [1..15] OF CHAR;                { מחרוזת }
   FULL_NAME = RECORD                                        { רשומת שם מלא }
                  FIRST_NAME,                                { שם פרטי }
                  FAMILY_NAME : CHR_STRING                   { שם משפחה }
               END;
   DATE = RECORD                                             { רשומת תאריך }
             DAY   : 1..31;                                  { יום }
             MONTH : 1..12;                                  { חודש }
             YEAR  : 1900..2000                              { שנה }
          END;
   STREET_REC = RECORD                                       { רשומת רחוב }
                   NAME   : CHR_STRING;                      { שם הרחוב }
                   NUMBER : 1..MAXINT                        { מספר הבית }
                END;
   ADDRESS_REC = RECORD                                      { רשומת עיר }
                    CITY,                                    { שם העיר }
                    ZIP_CODE : CHR_STRING;                   { מיקוד }
                    STREET   : STREET_REC                    { פרטי הרחוב }
                 END;
   STATUS = (RAVAK, NASUI, ALMAN, GARUSH, NASUI_2);          { מצב משפחתי }
```

```
NETUNIM = RECORD                                              { רשומת נתונים }
            NAME : FULL_NAME;                                 { פרטי השם     }
            ID   : REAL;                                      { תעודת זהות   }
            BORN_DATE : DATE;                                 { תאריך לידה   }
            CHILDREN  : 1..16;                                { מספר ילדים   }
            ADDRESS   : ADDRESS_REC;                          { פרטי הכתובת  }
            SEX : (MALE, FEMALE);                             { מין          }
            CASE  FAMILY_STATUS : STATUS  OF                  { מצב משפחתי   }
               RAVAK  : (HOUSE     : BOOLEAN); { רווק: יש דירה?        }
               NASUI  : (MAR_DATE  : DATE;     { נשוי: ת. נישואין      }
                         B_NAME    : FULL_NAME); { שם בן/ת הזוג        }
               GARUSH : (FREE_DATE : DATE); { גרוש: ת. גירושים         }
               ALMAN  : (FIN_DATE  : DATE); { אלמן:                    }
               NASUI_2 : (FIRST_FALL,       { נשוי בשנית:              }
                          END_FIRST,
                          SECOND_FALL  : DATE;
                          FIRST_B_NAME : FULL_NAME)
          END;
```



תוכניות

תוכנית 4

```
PROGRAM STOCK_FILES (OLD_STOCK, NEW_STOCK, TRANSACTIONS, OUTPUT);

TYPE
   POSITIVE_INTEGER = 0..MAXINT;
   ITEM = RECORD                                              { רשומת פריט }
             NUMBER,                                          { מספר פריט }
             IN_STOCK,                                        { מלאי קיים }
             TARGET_STOCK,                                    { רמת מלאי רצויה }
             ON_ORDER,                                        { כמות בהזמנה }
             REORDER_LIMIT : POSITIVE_INTEGER                 { נקודת הזמנה }
          END;
   STOCK_FILE = FILE OF ITEM;
   TRANS_ITEM = RECORD                                        { רשומת תנועה }
                   NUMBER,                                    { מספר פריט }
                   AMOUNT     : POSITIVE_INTEGER;             { כמות }
                   TRANS_TYPE : (DELIVERY, DISPATCH)          { סוג פעולה }
                END;
   TRANS_FILE = FILE OF TRANS_ITEM;
   RELATION = (SMALLER, EQUAL, BIGGER);
VAR
   OLD_STOCK,                                                 { קובץ מלאי ישן }
   NEW_STOCK           : STOCK_FILE;                          { קובץ מלאי חדש (מעודכן) }
   TRANSACTIONS        : TRANS_FILE;                          { קובץ תנועות }
   TRANS               : TRANS_ITEM;                          { רשומת תנועה }
   OLD_ITEM            : ITEM;                                { רשומת פריט }
   END_TRANS_FILE,
   END_STOCK_FILE      : BOOLEAN;

                                                   { פרוצדורה להדפסת קובץ המלאי }

   PROCEDURE PRINT_STOCK (VAR STOCK : STOCK_FILE);
      BEGIN
         RESET (STOCK);
         WRITELN;
         WRITELN
     ('     ITEM NUMBER   IN STOCK   ON ORDER    TARGET      REORDER');
         WHILE  NOT EOF (STOCK)  DO
          BEGIN
            READ (STOCK, OLD_ITEM);
            WITH  OLD_ITEM  DO
               WRITELN (NUMBER:12, IN_STOCK:12, ON_ORDER:12,
                        TARGET_STOCK:12, REORDER_LIMIT:12)
          END
    END; { print_stock }
```

```
                    { הפונקציה COMPARE משווה בין מספר פריט מקובץ המלאי
                    לבין מספר הפריט בתנועה, ומחזירה את היחס ביניהם. }

FUNCTION COMPARE (ITEM_NUMBER,
                  TRANS_NUMBER : POSITIVE_INTEGER) : RELATION;
  BEGIN
    IF  ITEM_NUMBER < TRANS_NUMBER  THEN
      COMPARE := SMALLER
    ELSE
      IF  ITEM_NUMBER > TRANS_NUMBER  THEN
        COMPARE := BIGGER
      ELSE
        COMPARE := EQUAL          { מספר הפריט מקובץ המלאי שווה }
  END; { compare }                {              למספר פריט בתנועה. }

                         { הפרוצדורה ERROR מדפיסה הודעת שגיאה }

PROCEDURE ERROR (TRANS_NUMBER : POSITIVE_INTEGER);
  BEGIN
    WRITELN ('TRANSACTION REJECTED FOR ITEM ', TRANS_NUMBER:1)
  END; { error }

     { הפרוצדורה UPDATE מעדכנת רשומת פריט בקובץ הישן (OLD_ITEM)
                                    לפי רשומת תנועה (TRANS). }

PROCEDURE UPDATE (VAR OLD_ITEM : ITEM;
                      TRANS    : TRANS_ITEM);
  BEGIN
    WITH  OLD_ITEM, TRANS  DO
      CASE  TRANS_TYPE  OF
        DELIVERY : BEGIN                              { הוספה למלאי }
                     IN_STOCK := IN_STOCK + AMOUNT;
                                              { עדכון הכמות במלאי }
                     ON_ORDER := ON_ORDER - AMOUNT
                                             { עדכון הכמות בהזמנה }
                   END;
        DISPATCH : IN_STOCK := IN_STOCK - AMOUNT
                                   { הוצאה מהמלאי, עדכון הכמות שבמלאי }
      END
  END; { update }
```

```
{ פרוצדורה לעדכון הכמות בהזמנה
  וכתיבת הרשומה המעודכנת לקובץ החדש. }

   PROCEDURE WRITE_ITEM (VAR NEW_ITEM : ITEM);
      BEGIN
         WITH  NEW_ITEM  DO
            IF  IN_STOCK + ON_ORDER < REORDER_LIMIT  THEN
             BEGIN
               WRITELN ('ORDER ',TARGET_STOCK - ON_ORDER - IN_STOCK:1,
                        ' OF ITEM ', NUMBER);
               ON_ORDER := TARGET_STOCK - IN_STOCK
             END;
            WRITE (NEW_STOCK, NEW_ITEM)
      END; { write_item }

BEGIN { main }
   ASSIGN (OLD_STOCK, 'STOCK.OLD');
   ASSIGN (NEW_STOCK, 'STOCK.NEW');
   ASSIGN (TRANSACTIONS, 'TRANS.DAT');
   PRINT_STOCK (OLD_STOCK);              { הדפסה של קובץ המלאי לפני העדכון }
   RESET (OLD_STOCK);
   RESET (TRANSACTIONS);
   REWRITE (NEW_STOCK);
   END_TRANS_FILE := EOF (TRANSACTIONS);
   END_STOCK_FILE := EOF (OLD_STOCK);
   IF  NOT  (END_STOCK_FILE  OR  END_TRANS_FILE) THEN
    BEGIN
      READ (OLD_STOCK, OLD_ITEM);
      READ (TRANSACTIONS, TRANS);
      REPEAT
         CASE  COMPARE (OLD_ITEM.NUMBER, TRANS.NUMBER)  OF
                { השוואה של שדה מספר פריט ברשומת פריט וברשומת תנועה }
            SMALLER : BEGIN                { מספר פריט קטן ממספר התנועה. }
                        WRITE_ITEM (OLD_ITEM);      { כתיבת רשומת פריט
                                                      לקובץ החדש. }
                        IF  NOT EOF (OLD_STOCK)  THEN
                          READ (OLD_STOCK, OLD_ITEM)  { קריאת רשומת
                                                        פריט הבאה. }
                        ELSE  { eof }               { סוף קובץ פריטים }
                          END_STOCK_FILE := TRUE
                      END;
            EQUAL   : BEGIN                { מספר תנועה שווה למספר פריט }
                        UPDATE (OLD_ITEM, TRANS); { עדכון רשומת פריט }
                        IF  NOT EOF (TRANSACTIONS)  THEN
                          READ (TRANSACTIONS, TRANS)      { קריאה של
                                                         התנועה הבאה. }
                        ELSE  { eof }               { סוף קובץ התנועות }
                          END_TRANS_FILE := TRUE
                      END;
```

```
            BIGGER   : BEGIN                    { מספר פריט גדול ממספר התנועה }
                         ERROR (TRANS.NUMBER);             { הדפסת שגיאה }
                         IF  NOT EOF (TRANSACTIONS)  THEN
                           READ (TRANSACTIONS, TRANS)          { קריאה של }
                                                          { התנועה הבאה. }
                         ELSE  { eof }                   { סוף קובץ תנועות }
                           END_TRANS_FILE := TRUE
                       END
          END                                                    { לולאה }
       UNTIL  END_TRANS_FILE OR END_STOCK_FILE;         { עד סוף אחד הקבצים. }
       IF  END_TRANS_FILE  THEN               { אם זהו סוף קובץ התנועות
          WRITE_ITEM (OLD_ITEM)               { יש לכתוב לקובץ החדש את רשומת
                                              { הפריט שנקראה קודם.
       ELSE { end of stock file }                         { סוף קובץ מלאי }
          ERROR (TRANS.NUMBER);             { הדפסת שגיאה, כי עדיין יש תנועות }
    END; { if }
    WHILE  NOT EOF (OLD_STOCK)  DO
     BEGIN                                { קובץ הפריטים עדיין לא הגיע לסופו. }
       READ (OLD_STOCK, OLD_ITEM);          { העברת הרשומות הנותרות מהקובץ }
       WRITE_ITEM (OLD_ITEM)                              { הישן לקובץ החדש. }
     END;
    WHILE  NOT EOF (TRANSACTIONS)  DO
     BEGIN      { קובץ התנועות עדיין לא הסתיים, קובץ המלאי כבר הסתיים }
       READ (TRANSACTIONS, TRANS);
       ERROR (TRANS.NUMBER)                                { הדפסת שגיאה }
     END;
    PRINT_STOCK (NEW_STOCK);                     . { הדפסת קובץ המלאי המעודכן }
    CLOSE (OLD_STOCK);
    CLOSE (NEW_STOCK);
    CLOSE (TRANSACTIONS)
 END.
```

תוכנית 8

בתוכנית זו כללנו מספר פונקציות ופרוצדורות מעבר לנדרש לפתרון התרגיל, כפי שנוסח בספר פסקל.
תוכניות שרות כלליות, אשר נדרשות בעת עבודה בקבצים מסוג אינדקס-סידרתי (Index Sequential). לדוגמה, את הפרוצדורה CREATE_INDEX לבניית טבלת אינדקס לקובץ סידרתי אפשר ליישם עם שינויים קלים לפי תוכנית היישום. הפרוצדורה SAVE_INDEX מיועדת לשמירת טבלת האינדקס בקובץ כדי שאפשר יהיה להשתמש בה בהרצה הבאה של התוכנית ולחסוך בנייה של טבלת אינדקס בכל הרצה.

פונקציות ופרוצדורות נוספות מסייעות לתוכניתן לעקוב אחר תוכן קובץ הנתונים וטבלת האינדקס במהלך הביצוע של התוכנית. לדוגמה, הפרוצדורה INDEX_LIST מדפיסה את תוכן טבלת האינדקס בכל שלב במהלך ביצוע התוכנית.

גוף התוכנית נבנה סביב משפט CASE, כדי לאפשר גמישות מירבית בתכנון התפריט הראשי של התוכנית.

```
PROGRAM BANK (INPUT, INDEX_F, ACCOUNTS, OUTPUT);
CONST
   MAX_ACCOUNT = 100;              { מספר כניסות מכסימלי בטבלת האינדקס }
   RECORDS = -1;
   DELETED_RECORDS = -2;
   DELETED = 'ZZZZZZZZZZ';            { סימון של רשומה לא פעילה בקובץ }
   MAX_ACC = 99999999999.0;
TYPE
   CHR_STRING = PACKED ARRAY [1..10] OF CHAR;
   ACCOUNT_RECORD = RECORD                                  { רשומת חשבון }
                     NAME    : CHR_STRING;                  {          שם }
                     NUMBER,                                { מספר חשבון  }
                     CREDIT,                                {      אשראי  }
                     BALANCE : REAL                         {       יתרה  }
                   END;
   INDEX_RECORD = RECORD                             {        רשומת אינדקס }
                   NUMBER : REAL;                    { מפתח - מספר חשבון  }
                   RECORD_NUMBER : INTEGER           {        מיקום בקובץ  }
                END;
   INDEX_TABLE = ARRAY [-2..MAX_ACCOUNT] OF INDEX_RECORD;

   ACC_FILE   = FILE OF ACCOUNT_RECORD;
   INDEX_FILE = FILE OF INDEX_RECORD;

VAR
   ACCOUNTS : ACC_FILE;                                  { קובץ החשבונות }
   INDEX_F  : INDEX_FILE;          { קובץ האינדקס בו נשמרת טבלת האינדקס }
   INDEX    : INDEX_TABLE;                                { טבלת האינדקס }
   ACCOUNT : ACCOUNT_RECORD;
   COMMAND : CHAR;
   ACC_NUMBER : REAL;
```

{ הפונקציה FIND_NUMBER מבצעת חיפוש בינרי בטבלת האינדקס, כדי למצוא מספר חשבון (NUMBER) נתון. החיפוש נעשה בתחום הרשומות הפעילות בלבד. אם נמצא מספר החשבון - הפונקציה תחזיר TRUE, אחרת - היא תחזיר FALSE. כאשר מספר החשבון נמצא בטבלת האינדקס, המשתנה MID מצביע על מיקומו, אחרת (מספר חשבון לא נמצא) - המשתנה MID מצביע על המקום בו הסתיים החיפוש. בערך MID נשתמש כדי למצוא את המיקום בטבלת האינדקס, אשר בה נכתוב מספר חשבון חדש. }

```
FUNCTION FIND_NUMBER (NUMBER  : REAL;
                      VAR MID : INTEGER) : BOOLEAN;
  VAR
    FIRST, LAST : INTEGER;
  BEGIN
    FIRST := 0;   { המיקום של הרשומה הפעילה הראשונה בטבלת האינדקס. }
    LAST := INDEX [RECORDS].RECORD_NUMBER;  { המיקום של הרשומה
                                             הפעילה האחרונה בטבלת האינדקס. }
    REPEAT                                  { חיפוש בינרי }
      MID := (FIRST + LAST) DIV 2;
      IF  INDEX [MID].NUMBER > NUMBER  THEN
        LAST := MID - 1
      ELSE
        IF  INDEX [MID].NUMBER < NUMBER  THEN
          FIRST := MID + 1
    UNTIL  (INDEX [MID].NUMBER = NUMBER) OR (LAST < FIRST);
    FIND_NUMBER := INDEX [MID].NUMBER = NUMBER
  END;  { find_number }
```

{ הפונקציה LAST_INDEX_RECORD מחזירה את המיקום בטבלת האינדקס של הרשומה הפעילה האחרונה. שים לב, שמספר זה קטן ב-1 ממספר הרשומות הפעילות, כי הרשומה הפעילה הראשונה נמצאת במקום 0 בטבלה. }

```
FUNCTION LAST_INDEX_RECORD : INTEGER;
  BEGIN
    LAST_INDEX_RECORD := INDEX [RECORDS].RECORD_NUMBER
  END;  { last_index_record }
```

{ הפונקציה FIND_FREE_SPACE מוצאת מקום פנוי בקובץ הפיסי כדי לכתוב בו את הרשומה החדשה, ומחזירה אותו במשתנה FREE. הפונקציה תחזיר ערך TRUE אם גודל טבלת האינדקס מאפשר הוספת רשומה, או FALSE - אם טבלת האינדקס מלאה.
סדר העדיפות במציאת המיקום לרשומה החדשה הוא:
א. מיקום של רשומה המסומנת לא-פעילה (במקרה זה הפונקציה מעדכנת את מספר הרשומות הלא-פעילות),
ב. אם אין רשומות לא-פעילות בקובץ, מיקום הרשומה החדשה יהיה בסוף הקובץ. }

```
FUNCTION FIND_FREE_SPACE (VAR FREE  : INTEGER) : BOOLEAN;
   BEGIN
      FREE := MAX_ACCOUNT + 1;
      IF  INDEX [DELETED_RECORDS].RECORD_NUMBER > 0  THEN
       BEGIN                                 { יש רשומות לא-פעילות בקובץ. }
         FREE := INDEX [LAST_INDEX_RECORD + 1].RECORD_NUMBER; {מיקום }
                                             { של רשומה לא-פעילה בקובץ. }
         WITH  INDEX [DELETED_RECORDS]  DO
            RECORD_NUMBER := RECORD_NUMBER - 1
       END
      ELSE                                   { אין רשומות לא-פעילות בקובץ. }
         WITH  INDEX [RECORDS]  DO
            IF  RECORD_NUMBER < MAX_ACCOUNT  THEN    { יש מקום בטבלה. }
               FREE := RECORD_NUMBER + 1   { מיקום הרשומה בסוף הקובץ. }
            ELSE                           { אין מקום בטבלת האינדקס. }
               WRITELN ('ERROR! - OUT OF SPACE (MAX_ACCOUNT)');
      FIND_FREE_SPACE := FREE <= MAX_ACCOUNT
   END;  { find_free_space }

{ הפרוצדורה INSERT_TO_INDEX מכניסה רשומה חדשה לטבלת האינדקס
  לפי סדר מפתח המיון, שהוא מספר החשבון.
  FILE_LOCATION - מיקום הרשומה בקובץ.
  ACC_NUMBER    - מספר חשבון (מפתח).
  LOCATION      - הערך שמוחזר ע"י הפונקציה FIND_NUMBER במשתנה MID,
                  כאשר המספר לא נמצא בטבלה. מיקום הרשומה החדשה יכול
                  להיות במקום MID, או במקום שאחריו.
  לאחר הוספת הרשומה בטבלת האינדקס, מתבצע עדכון של מספר הרשומה
  הפעילה האחרונה בטבלה זו. }

PROCEDURE INSERT_TO_INDEX (FILE_LOCATION,
                           LOCATION    : INTEGER;
                           ACC_NUMBER : REAL);
  VAR
     L : INTEGER;
  BEGIN
     IF  INDEX [LOCATION].NUMBER < ACC_NUMBER  THEN
        LOCATION := LOCATION + 1;               { מציאת המיקום המדוייק }
                                                {  להכנסת הרשומה.     }
     FOR  L := LAST_INDEX_RECORD  DOWNTO  LOCATION  DO
        INDEX [L + 1] := INDEX [L];                        { פינוי מקום }
     INDEX [LOCATION].NUMBER := ACC_NUMBER;                    { מפתח }
     INDEX [LOCATION].RECORD_NUMBER := FILE_LOCATION;          { מיקום }
                                                    {  הרשומה בקובץ. }
     INDEX [RECORDS].RECORD_NUMBER :=               { עדכון מספר הרשומה }
           INDEX [RECORDS].RECORD_NUMBER + 1        {  הפעילה האחרונה.  }
  END;  { insert_to_index }
```

{ הפונקציה RECORD_DELETED מחזירה TRUE אם הרשומה ACCOUNT אינה-פעילה,
אחרת - היא תחזיר FALSE. }

```
FUNCTION RECORD_DELETED (ACCOUNT : ACCOUNT_RECORD) : BOOLEAN;
   BEGIN
      RECORD_DELETED := ACCOUNT.NAME = DELETED
   END;  { record_deleted }
```

{ הפרוצדורה CREATE_INDEX בונה טבלת אינדקס לקובץ החשבונות.
שדה המפתח הוא שדה מספר החשבון.
הערה: לשדה המפתח ברשומה במיקום 0 ניתן ערך גדול ביותר,
כדי שהוספת הרשומה הראשונה תהיה במקום המתאים.
מבנה טבלת האינדקס: בראש הטבלה נמצאות הרשומות הפעילות, אשר
ממויינות על פי שדה המפתח (מספר חשבון). הכנסת הרשומה למקומה בטבלה
נעשה ע"י שיבוץ. אחרי הרשומות הפעילות תהיינה הרשומות שאינן פעילות. }

```
PROCEDURE CREATE_INDEX;
   VAR
      ACCOUNT : ACCOUNT_RECORD;
      FILE_LOCATION,
      LOCATION       : INTEGER;
   BEGIN
      INDEX [DELETED_RECORDS].RECORD_NUMBER := 0;          { אין רשומות  }
                                                           { לא-פעילות. }
      INDEX [RECORDS].RECORD_NUMBER := -1;             { אין רשומות פעילות }
      INDEX [0].NUMBER := MAX_ACC;
      RESET (ACCOUNTS);
      FILE_LOCATION := 0;
      WHILE  NOT EOF (ACCOUNTS)  DO
       BEGIN
         READ (ACCOUNTS, ACCOUNT);
         IF  RECORD_DELETED (ACCOUNT)  THEN          { אם הרשומה לא-פעילה }
          BEGIN
            WITH  INDEX [DELETED_RECORDS]  DO          { עדכון מספר הרשומות }
               RECORD_NUMBER := RECORD_NUMBER + 1;   {  שאינן-פעילות. }
            INDEX [FILE_LOCATION].RECORD_NUMBER := FILE_LOCATION
          END          { הוספה לאיזור הרשומות הלא-פעילות בטבלת האינדקס }
         ELSE                                         { אחרת - רשומה פעילה. }
                               { אם המפתח מופיע בטבלת האינדקס זוהי שגיאה }
            IF  FIND_NUMBER (ACCOUNT.NUMBER, LOCATION)  THEN
               WRITELN ('ERROR! - DUPLICATE ACCOUNT NUMBER')
            ELSE                                  { אחרת - המפתח לא מופיע }
             BEGIN                              { פינוי מקום בטבלת האינדקס }
               INDEX [FILE_LOCATION] := INDEX [LAST_INDEX_RECORD + 1];
               INSERT_TO_INDEX        { הוספת כניסה בטבלה לרשומה החדשה }
                          (FILE_LOCATION, LOCATION, ACCOUNT.NUMBER)
             END;  { else }
         FILE_LOCATION := FILE_LOCATION + 1                   { מיקום הרשומה }
       END   { while }                                         {  הבאה בקובץ. }
   END;  { create_index_file }
```

```
{ הפרוצדורה APPEND בודקת אם רשומת החשבון (ACCOUNT) נמצאת בקובץ.
  אם הרשומה לא נמצאת בקובץ, היא מוסיפה אותה לקובץ ומעדכנת
  את טבלת האינדקס. }

PROCEDURE APPEND (VAR ACCOUNT : ACCOUNT_RECORD);
  VAR
    FILE_LOCATION,
    LOCATION     : INTEGER;
  BEGIN                       { אם המפתח נמצא בטבלת האינדקס, המשמעות היא }
                              { שקיימת רשומת חשבון בקובץ, ולכן זו שגיאה. }
    IF  FIND_NUMBER (ACCOUNT.NUMBER, LOCATION)  THEN
      WRITELN ('ERROR! - DUPLICATE ACCOUNT NUMBER')
    ELSE                      { אחרת (לא נמצא מפתח) - לא קיימת רשומה כזו. }
      IF  FIND_FREE_SPACE (FILE_LOCATION) THEN  { אם יש מקום פנוי }
       BEGIN
         SEEK (ACCOUNTS, FILE_LOCATION);        { מיקום הרשומה בקובץ }
         WRITE (ACCOUNTS, ACCOUNT);       { כתיבת הרשומה לקובץ הפיסי }
         INSERT_TO_INDEX (FILE_LOCATION, LOCATION, ACCOUNT.NUMBER)
       END                                      { עדכון של טבלת האינדקס }
  END;  { append }

{ הפרוצדורה DELETE מסמנת רשומת חשבון כרשומה לא-פעילה, על פי מספר
  החשבון (ACC_NUMBER) שהיא מקבלת. הפרוצדורה מעבירה את רשומת
  האינדקס של הרשומה שנמחקה מהאיזור הפעיל לאיזור הלא-פעיל בטבלת
  האינדקס. בתוך כך, היא מעדכנת את מספר הרשומות הפעילות והלא-פעילות. }

PROCEDURE DELETE (ACC_NUMBER : REAL);
  VAR
    LOCATION,                  { מיקום הכניסה של הרשומה בטבלת האינדקס }
    L         : INTEGER;
    TEMP_INDEX  : INDEX_RECORD;
    ACCOUNT : ACCOUNT_RECORD;
  BEGIN
    IF  FIND_NUMBER (ACC_NUMBER, LOCATION)  THEN            { אם נמצאה }
     BEGIN                                           { הרשומה המבוקשת. }
      SEEK (ACCOUNTS,INDEX [LOCATION].RECORD_NUMBER); {מיקום בקובץ}
      ACCOUNT.NAME := DELETED;               { סימון הרשומה כלא-פעילה. }
      WRITE (ACCOUNTS, ACCOUNT);
      TEMP_INDEX := INDEX [LOCATION];        { הכניסה בטבלת האינדקס }
                                             { של הרשומה הלא-פעילה. }
      FOR  L := LOCATION  TO  LAST_INDEX_RECORD - 1  DO
        INDEX [L] := INDEX [L + 1];
      INDEX [LAST_INDEX_RECORD] := TEMP_INDEX;
      WITH  INDEX [RECORDS]  DO  { עדכון המצביע לרשומות הפעילות. }
        RECORD_NUMBER := RECORD_NUMBER - 1;
      WITH  INDEX [DELETED_RECORDS]  DO     { עדכון מספר הרשומות }
        RECORD_NUMBER := RECORD_NUMBER + 1  {         שאינן פעילות. }
    END  { if }
   ELSE                               { אחרת - הרשומה לא נמצאה }
     WRITELN ('ERROR! - ACCOUNT NUMBER NOT FOUND')        { שגיאה }
 END;  { delete }
```

```
{      הפרוצדורה UPDATE מאפשרת עדכון פרטים של רשומת חשבון שמספרה
ACC_NUMBER. הגישה לקובץ מתבצעת לפי טבלת האינדקס.
אם לא נמצאה רשומה בעלת מספר חשבון כזה מוצגת הודעת שגיאה.
הערה: עדכון מספר חשבון חייב להתבצע ע"י מחיקת רשומת חשבון והוספה
של רשומה עם המספר החדש. }

PROCEDURE UPDATE (ACC_NUMBER : REAL);
   VAR
      LOCATION : INTEGER;      { מיקום הכניסה של הרשומה בטבלת האינדקס. }
      ACCOUNT  : ACCOUNT_RECORD;
   BEGIN
      IF  FIND_NUMBER (ACC_NUMBER, LOCATION)  THEN
       BEGIN                                          { אם הרשומה נמצאה }
         SEEK (ACCOUNTS, INDEX [LOCATION].RECORD_NUMBER);
         READ (ACCOUNTS, ACCOUNT);                    { קריאת הרשומה }
         WRITELN ('ACCOUNTS DETAILS ARE:');
         WRITELN('NAME ':10, 'ACCOUNT ':15, 'CREDIT':8, 'BALANCE':11);
         WITH  ACCOUNT  DO                    { קריאה של נתונים חדשים }
            WRITELN (NAME:15, NUMBER:8:0, CREDIT:10:2, BALANCE:10:2);
         WRITELN ('ENTER NEW DETAILS:');
         WRITELN ('NAME ':10, 'CREDIT':13, 'BALANCE':11);
         WITH  ACCOUNT  DO
            READLN (NAME, CREDIT, BALANCE);
         SEEK (ACCOUNTS, INDEX [LOCATION].RECORD_NUMBER);
         WRITE (ACCOUNTS, ACCOUNT)        { כתיבת הפרטים החדשים בקובץ }
       END
      ELSE                                          { הרשומה לא נמצאה }
         WRITELN ('ERROR! - ACCOUNT NUMBER NOT FOUND')
   END;    { update }

{    הפרוצדורה REARRANGE_FILE מארגנת מחדש לפי סדר המיון את רשומות
החשבון הפעילות בלבד, ובונה טבלת אינדקס חדשה.
הערה: הפרוצדורה ERASE אינה שייכת לפסקל הסטנדרטי ומיושמת במהדרים
שונים בשמות שונים. }

PROCEDURE  REARRANGE_FILE;
   VAR
      I : INTEGER;
      TEMP : ACC_FILE;
      ACCOUNT : ACCOUNT_RECORD;
BEGIN
   WRITELN ('ARRANG ACCOUNTS FILE');
   ASSIGN (TEMP, 'TEMP.ACC');
   REWRITE (TEMP);
   FOR  I := 0  TO  LAST_INDEX_RECORD  DO        { מעבר על כל הרשומות }
      WITH  INDEX [I]  DO                 {  לפי סדר המיון בטבלת האינדקס. }
       BEGIN
         SEEK (ACCOUNTS, RECORD_NUMBER);                   { מיקום הרשומה }
         READ (ACCOUNTS, ACCOUNT);           { קריאת הרשומה מקובץ המקור }
         WRITE (TEMP, ACCOUNT)                { כתיבת הרשומה לקובץ זמני }
```

```
      END;
   REWRITE (ACCOUNTS);                          { איתחול הקובץ המקורי }
   RESET (TEMP);                                { קובץ TEMP ממויין לפי מפתח ומכיל }
   WHILE  NOT EOF (TEMP)  DO                    { רשומות פעילות בלבד. }
    BEGIN                                       { העתקת קובץ }
      READ (TEMP, ACCOUNT);                     { קריאת רשומה מקובץ TEMP }
      WRITE (ACCOUNTS, ACCOUNT)                 { וכתיבתה בקובץ ACCOUNTS. }
    END;
   ERASE (TEMP);                                { מחיקת הקובץ הזמני. }
   CREATE_INDEX                                 { יצירת טבלת אינדקס חדשה }
END;  { rearrange_file }

PROCEDURE SAVE_INDEX;                           { שמירת טבלת האינדקס בקובץ }
   VAR
      L : INTEGER;
   BEGIN
      WRITELN ('AVE INDEX TABLE IN FILE');
      REWRITE (INDEX_F);
      FOR  L := -2  TO  INDEX [RECORDS].RECORD_NUMBER +
                        INDEX [DELETED_RECORDS].RECORD_NUMBER  DO
         WRITE (INDEX_F, INDEX [L])
   END;  { save_index }

PROCEDURE  LOAD_INDEX;                          { טעינת טבלת האינדקס מקובץ }
   VAR
      L : INTEGER;
   BEGIN
      WRITELN ('OAD INDEX TABLE FROM FILE');
      RESET (INDEX_F);
      L := -2;
      WHILE  NOT EOF (INDEX_F)  DO
       BEGIN
         READ (INDEX_F, INDEX [L]);
         L := L + 1
       END
   END;  { load_index }

                                    { הפרוצדורה READ_ACCOUNT קוראת נתוני
                                      חשבון אל הרשומה ACCOUNT. }

PROCEDURE READ_ACCOUNT (VAR ACCOUNT : ACCOUNT_RECORD);
   BEGIN
      WRITELN ('     NAME        NUMBER    CREDIT    BALANCE');
      WITH  ACCOUNT  DO
         READLN (NAME, NUMBER, CREDIT, BALANCE)
   END;  { read_account }
```

```
PROCEDURE PRINT_ACCOUNTS;  { הדפסה של הרשומות הפעילות בקובץ, לפי סדר }
   VAR                     { המיון, שקיים בטבלת האינדקס.            }
      I : INTEGER;
   BEGIN
      WRITELN ('RINT ACCOUNTS LIST.');
      WRITELN;
      WRITELN ('NAME ':10, 'ACCOUNT ':15, 'CREDIT':8, 'BALANCE':11);
      WRITELN ('  ==============================================');
      FOR I := 0  TO  LAST_INDEX_RECORD  DO         { לולאה על הכניסות }
         WITH  INDEX [I]  DO                        { הפעילות.          }
          BEGIN
            SEEK (ACCOUNTS, RECORD_NUMBER);               { מיקום בקובץ }
            READ (ACCOUNTS, ACCOUNT);
            WITH  ACCOUNT  DO
              WRITELN (NAME:15, NUMBER:8:0, CREDIT:10:2, BALANCE:10:2)
          END
   END;  { print_accounts }

                           { הדפסת הרשומות לפי הסדר שהן נמצאות בקובץ
                             - רשומות פעילות ולא-פעילות (אם ישנן). }

PROCEDURE FILE_LIST;
   BEGIN
      WRITELN ('ILE LIST');
      WRITELN ('NAME ':10, 'ACCOUNT ':15, 'CREDIT':8, 'BALANCE':11);
      WRITELN ('  ==============================================');
      RESET (ACCOUNTS);
      WHILE  NOT EOF (ACCOUNTS)  DO
       BEGIN
         READ (ACCOUNTS, ACCOUNT);
         WITH  ACCOUNT  DO
            WRITELN (NAME:15, NUMBER:8:0, CREDIT:10:2, BALANCE:10:2)
       END
   END;  { file_list }

PROCEDURE INDEX_LIST;                          { הדפסה של טבלת האינדקס }
   VAR
      L : INTEGER;
   BEGIN
      WRITELN ('NDEX LIST');
      WRITELN ('INDEX    ACCOUNT  RECORD NUMBER');
      FOR  L := -2  TO  INDEX [RECORDS].RECORD_NUMBER +
                          INDEX [DELETED_RECORDS].RECORD_NUMBER  DO
         WITH  INDEX [L]  DO
            WRITELN (L:4, NUMBER:12:0, RECORD_NUMBER:10)
   END;  { index_list }

BEGIN  { main }
   ASSIGN (ACCOUNTS, 'ACCOUNTS.DTA');
   ASSIGN (INDEX_F, 'ACCOUNTS.NDX');
   RESET (ACCOUNTS);
```

```
REPEAT
  WRITELN;        { .הרשימה הבאה כוללת את הפעולות שהמשתמש מורשה לבצע }
                  { .עליו להקיש את האות המציינת את הפעולה }
  WRITELN ('Print / Delete / Append / Update /              Quit');
  WRITELN (' File list / Rearrange file');
  WRITELN (' Index list / Create index / Save index / Load index');
  WRITELN;
  WRITE ('COMMAND? ');
  READ (COMMAND);
  CASE  COMMAND  OF
     'A' : BEGIN                              { הוספה של רשומת חשבון חדשה }
              WRITELN ('PPEND NEW ACCOUNT');
              READ_ACCOUNT (ACCOUNT);
              APPEND (ACCOUNT)
           END;
     'C' : BEGIN                              { בנייה של טבלת האינדקס }
              WRITELN ('REATE INDEX TABLE');
              CREATE_INDEX
           END;
     'D' : BEGIN                              { מחיקה של רשומת חשבון }
              WRITE ('ELETE - ENTER ACCOUNT NUMBER: ');
              READLN (ACC_NUMBER);
              WRITELN;
              DELETE (ACC_NUMBER)
           END;
     'F' : FILE_LIST;         { .הדפסת רשומות לפי הסדר שלהן בקובץ }
     'I' : INDEX_LIST;                  { .הדפסת טבלת אינדקס }
     'L' : LOAD_INDEX;                  { .טעינת טבלת אינדקס מקובץ }
     'P' : PRINT_ACCOUNTS;              { .הדפסת רשומות לפי סדר המיון }
     'Q' : WRITELN ('UIT');             { .יציאה מהתוכנית - סיום }
     'R' : REARRANGE_FILE;     { .ארגון מחדש של הקובץ וטבלת האינדקס }
     'S' : SAVE_INDEX;                  { .שמירת טבלת האינדקס בקובץ }
     'U' : BEGIN                        { .עדכון פרטי רשומת חשבון }
             WRITE ('PDATE - ENTER ACCOUNT NUMBER: ');
             READLN (ACC_NUMBER);
             WRITELN;
             UPDATE (ACC_NUMBER)
           END
  END    { case }
UNTIL  COMMAND = 'Q';
CLOSE (ACCOUNTS);
CLOSE (INDEX_F)
END.
```

תיאור סכימטי של הרצת התוכנית

תיאור קובץ החשבונות:

מספר רשומה	NAME	(שדה מפתח) NUMBER	CREDIT	BALANCE
0	HADAR	78211	7000.0	31149.22
1	URI	54590	17000.0	142000.0
2	ZZZZZZZZZZ	12345	2000.0	20000.0
3	AVIV	33494	4000.0	14966.5
4	ZZZZZZZZZZ	99461	7000.0	31149.22
5	OREN	23146	1400.0	81136.3

טבלת האינדקס, אשר מציינת את הרשומות בקובץ:

אינדקס האיבר בטבלה (המערך)	שדה מפתח	מספר רשומה	
-2		2	מספר הרשומות הלא-פעילות
-1		3	מיקום הרשומה הפעילה האחרונה בטבלת האינדקס (*)
0	23146	5	כניסות של רשומות פעילות
1	33494	3	
2	54590	1	
3	78211	0	
4	99461	4	כניסות של רשומות שאינן פעילות
5	12345	2	

(*) מספר זה קטן ב-1 ממספר הרשומות הפעילות.

בשני המקומות הראשונים בטבלת האינדקס בשדה RECORD_NUMBER מוחזקים הנתונים הבאים:

א. במקום 2- נמצא מספר המציין את מספר הרשומות שאינן פעילות בקובץ. בפועל, רשומות אלו אינן נמחקות מהקובץ, אלא רושמים את הערך 'ZZZZZZZZZZ' בשדה השם שלהן. ברשומות אלו ניתן להשתמש שוב, אם דרוש.

ב. במקום 1- נמצא מספר המצביע על המיקום של הרשומה הפעילה האחרונה בטבלת האינדקס. מספר זה קטן ב-1 ממספר הרשומות הפעילות, מכיוון שהרשומה הראשונה נמצאת במקום 0. תוכל להבחין בדוגמה, שמספר הרשומות הפעילות הוא 4, והמספר הנמצא במקום 1- בטבלה הוא 3.

שדה מספר הרשומה מציין את מספר הרשומה בקובץ. הרשומה הראשונה בקובץ היא רשומה 0.

שדות המפתח (מספרי חשבון) של הרשומות הפעילות בטבלת האינדקס רשומות לפי סדר המיון.



דוגמה לביצוע הפרוצדורה DELETE:

בדוגמה זו, רוצים לבטל את מספר החשבון 54590, אשר נמצא הן בקובץ והן בטבלת האינדקס. נבצע זאת ע"י הפקודה DELETE(54590).
- הפרוצדורה FIND_NUMBER מבצעת חיפוש בינרי על טבלת האינדקס, מוצאת את מספר החשבון ומחזירה TRUE. המשתנה LOCATION יצביע על כניסה 2 בטבלת האינדקס. שדה מספר הרשומה בכניסה זו מורה שמספר הסידורי של הרשומה בקובץ הוא 1.
- נבצע גישה לרשומה מספר 1 בקובץ ע"י הפרוצדורה SEEK, ונסמן את הרשומה כלא פעילה.
- נעביר את כניסה מספר 2 בטבלת האינדקס מאיזור הרשומות הפעילות לאיזור הרשומות שאינן פעילות.
- נעדכן בטבלת האינדקס את המונה של הכניסות (הרשומות) הפעילות ואת המיקום של הכניסה (הרשומה) הפעילה האחרונה.

תיאור סכימטי של קובץ הנתונים וטבלת האינדקס לאחר ביצוע DELETE.

תיאור קובץ החשבונות:

מספר רשומה	NAME	(שדה מפתח) NUMBER	CREDIT	BALANCE
0	HADAR	78211	7000.0	31149.22
1	ZZZZZZZZZZ	54590	17000.0	142000.0
2	ZZZZZZZZZZ	12345	2000.0	20000.0
3	AVIV	33494	4000.0	14966.5
4	ZZZZZZZZZZ	99461	7000.0	31149.22
5	OREN	23146	1400.0	81136.3

מצב טבלת האינדקס, לאחר ביטול מספר חשבון 54590:

אינדקס האיבר בטבלה (המערך)	שדה מפתח	מספר רשומה	
-2		3	מספר הרשומות הלא-פעילות
-1		2	מיקום הרשומה הפעילה האחרונה בטבלת האינדקס
0	23146	5	כניסות של רשומות פעילות
1	33494	3	כניסות של רשומות פעילות
2	78211	0	כניסות של רשומות פעילות
3	54590	1	כניסות של רשומות שאינן פעילות
4	99461	4	כניסות של רשומות שאינן פעילות
5	12345	2	כניסות של רשומות שאינן פעילות

דוגמה לביצוע הפרוצדורה APPEND.

נתוני הרשומה החדשה:

NAME	(שדה מפתח) NUMBER	CREDIT	BALANCE
ZOHAR	28467	1000	1500

לאחר ביצוע הפרוצדורה FIND_NUMBER, המשתנה LOCATION יצביע על הכניסה מספר 1 בטבלת האינדקס. הפונקציה FIND_FREE_SPACE מחזירה במשתנה FREE את הערך 1. כלומר, במקום זה בקובץ נמצאת רשומה לא פעילה וניתן לכתוב במקומה רשומה חדשה. הפונקציה FIND_FREE_SPACE מעדכנת את מספר הרשומות שאינן פעילות.

- הרשומה החדשה נכתבת למיקום 1 בקובץ.

- עדכון טבלת האינדקס ע"י הפרוצדורה INSERT_TO_INDEX:
  א. ערכו של LOCATION לאחר ביצוע FIND_NUMBER הוא 1. ערך המפתח בכניסה 1 בטבלת האינדקס גדול מערך המפתח של הרשומה החדשה, ולכן LOCATION נשאר 1.
  ב. הזזה של הכניסות של הרשומות הפעילות ממיקום LOCATION והלאה, מקום אחד קדימה. הדבר גורם למחיקת כניסה מספר 3, אשר מציינת שהרשומה במקום 1 בקובץ אינה פעילה.
  ג. שדה המפתח בכניסה 1 בטבלה מקבל את המפתח של הרשומה החדשה.
  ד. מתעדכן המספר של הרשומה הפעילה האחרונה בטבלת האינדקס (גדל ב-1).

תיאור קובץ החשבונות לאחר הוספת רשומה חדשה:

מספר רשומה	NAME	(שדה מפתח) NUMBER	CREDIT	BALANCE
0	HADAR	78211	7000.0	31149.22
1	ZOHAR	28467	1000.0	1500.0
2	ZZZZZZZZZZ	12345	2000.0	20000.0
3	AVIV	33494	4000.0	14966.5
4	ZZZZZZZZZZ	99461	7000.0	31149.22
5	OREN	23146	1400.0	81136.3

טבלת האינדקס, לאחר העדכון:

מספר רשומה	שדה מפתח	אינדקס האיבר בטבלה (המערך)	
2		-2	מספר הרשומות הלא-פעילות
3		-1	מיקום הרשומה הפעילה האחרונה בטבלת האינדקס
5	23146	0	כניסות של רשומות פעילות
1	28467	1	
3	33494	2	
0	78211	3	
4	99461	4	כניסות של רשומות שאינן פעילות
2	12345	5	

<u>הערה:</u> שדה המפתח בכניסות של הרשומות שאינן פעילות בטבלת האינדקס אינו דרוש.

פרק 13
קבצי תמליל

תוכניות

תוכנית 1

```
PROGRAM READ_TEXT (INPUT, TEXT_FILE);
CONST
   LINE_LENGTH = 70;                          { אורך שורה מכסימלי }
TYPE
   TEXT_LINE = PACKED ARRAY [1..LINE_LENGTH] OF CHAR;
VAR
   TEXT_FILE : TEXT;                          { קובץ התמליל }
   LINE : TEXT_LINE;                          { שורה }
   I,
   END_LINE : INTEGER;
BEGIN
   ASSIGN (TEXT_FILE, 'TEXT.DAT');
   REWRITE (TEXT_FILE);
   READLN (LINE);                             { קריאה של השורה הראשונה }
   WRITE (TEXT_FILE, '  ');                   { רווח של התחלת פיסקה }
   WHILE  LINE [1] <> '.'  DO                 { הלולאה מתבצעת עד סימן סוף הקלט }
    BEGIN                                     { - נקודה בתחילת שורה. }
      END_LINE := LINE_LENGTH;
      WHILE  LINE [END_LINE] = ' ' DO         { סריקה למציאת תו אחרון בשורה, }
         END_LINE := END_LINE - 1;            { שאינו רווח. }
      FOR  I := 1  TO  END_LINE  DO           { לולאה עד סוף השורה }
       BEGIN
         WRITE (TEXT_FILE, LINE [I]);         { כתיבת תו }
         IF  LINE [I] = '.'  THEN
          BEGIN                               { אם התו הוא נקודה }
            WRITE (TEXT_FILE, ' ');
            WHILE  (I < END_LINE) AND (LINE [I + 1] = ' ')  DO
```

```
            I := I + 1                                    { צמצום רווחים אחרי הנקודה }
         END
       END;    { for }
      WRITELN (TEXT_FILE);
      IF  (END_LINE < LINE_LENGTH) AND (LINE [END_LINE] = '.')  THEN
         WRITE (TEXT_FILE, '  ');  { אם אורך שורה (מיקום הנקודה) קטן }
                                   { מאורך שורה מכסימלי - זוהי פיסקה חדשה. }
      READLN (LINE)                                       { קריאת השורה הבאה }
    END;   { while }
   CLOSE (TEXT_FILE)
END.
```

תוכנית 3

```
PROGRAM FIND_TEXT (TEXT_FILE, INPUT, OUTPUT);
CONST
   MAX_WORD_LENGTH = 25;                              { אורך שורה מכסימלי }
TYPE
   WORD_TYPE = PACKED ARRAY [1..MAX_WORD_LENGTH] OF CHAR;
   QUEUE = RECORD                                     {          רשומת תור }
             FIRST,                                   { מצביע לראש התור  }
             LAST       : 0..20;                      { מצביע לזנב התור  }
             CHARACTER : ARRAY [0..20] OF CHAR       {       מערך תווים  }
          END;
VAR
   TEXT_FILE : TEXT;                                  { קובץ תמליל }
   CH        : CHAR;
   WORD,
   TEMP_WORD : WORD_TYPE;
   Q         : QUEUE;
   I,
   FOUND_COUNTER,              { מונה את מספר הפעמים שהמלה מופיעה בתמליל }
   WORD_LENGTH,
   WORD_LOCATION,
   CHAR_COUNTER    : INTEGER;

PROCEDURE MAKE_NULL (VAR Q : QUEUE);                  { איפוס התור }
  BEGIN
     WITH  Q  DO
      BEGIN
        FIRST := 0;
        LAST  := 0
      END
  END;     { nake_null }
```

```
PROCEDURE ADD_TO_QUEUE (VAR Q  : QUEUE;          { הוספה של התו CH לראש }
                             CH : CHAR);         { התור Q. }
   BEGIN
      WITH  Q  DO
       BEGIN
         CHARACTER [FIRST] := CH;                { הוספה לראש התור }
         FIRST := (FIRST + 1) MOD 21             { קידום המצביע לראש התור }
       END
   END;     { add_to_queue }

PROCEDURE SHIFT_QUEUE (VAR Q  : QUEUE;           { הכנסת התו CH לראש התור Q }
                           CH : CHAR);           { והשמטה של תו מזנב התור. }
   BEGIN
      WITH  Q  DO
       BEGIN
         CHARACTER [FIRST] := CH;
         FIRST := (FIRST + 1) MOD 21;
         LAST := (LAST + 1) MOD 21
       END
   END;     { shift_queue }

PROCEDURE SAVE_IN_Q (VAR Q  : QUEUE;   { שמירה של התו CH בתור Q, כאשר }
                         CH : CHAR;    { CHAR_COUNTER מונה את מספר }
                VAR CHAR_COUNTER : INTEGER);     { התווים בתור. }
   BEGIN
      CHAR_COUNTER := CHAR_COUNTER + 1;
      IF CHAR_COUNTER <= 20  THEN               { אם אין עדיין 20 תווים בתור }
         ADD_TO_QUEUE (Q, CH)                   { הוסף את התו CH לתור Q. }
      ELSE                                      { אחרת - ישנם כבר 20 תווים בתור. }
         SHIFT_QUEUE (C, CH)                    { הוסף את התו CH בראש התור Q והשמט }
   END;     { save_in_q }                       { תו מזנב התור. }

PROCEDURE PRINT_QUEUE (VAR Q : QUEUE);   { הדפסת תווים בתור Q לפי סדר }
   VAR                                   { כניסתם. כלומר, מזנב התור כלפי ראשו }
      I : INTEGER;
   BEGIN
      WITH  Q  DO
       BEGIN
         I := LAST;
         WHILE  I <> FIRST  DO
          BEGIN
            WRITE (CHARACTER [I]);
            I := (I + 1) MOD 21
          END
      END    { with }
   END;     { print_queue }
```

```
FUNCTION EQUAL (VAR WORD,
                    TEMP_WORD : WORD_TYPE;  { הפונקציה מחזירה TRUE אם }
                    LENGTH    : INTEGER   ) : BOOLEAN;  { שתי המחרוזות שוות. }
  VAR
    I : INTEGER;
  BEGIN
    I := 1;
    WHILE (I <= LENGTH) AND (WORD [I] = TEMP_WORD [I])  DO {השוואת}
        I := I + 1;                                      { תוים. }
    EQUAL := (I > LENGTH) AND (LENGTH = WORD_LENGTH)     { השוואת }
  END;    { wqual }                                      { אורכי המחרוזות. }

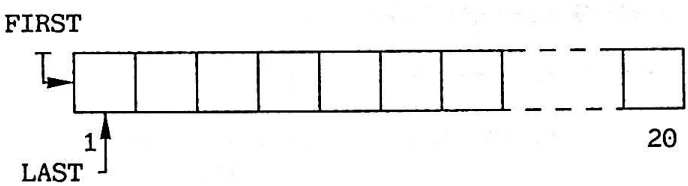
FUNCTION ALPHA_NUM (CH : CHAR) : BOOLEAN;    { הפונקציה מחזירה TRUE }
  BEGIN           { אם התו CH הוא אלפאנומרי (אות לועזית או ספרה) }
    ALPHA_NUM := (CH >= 'A') AND (CH <= 'Z') OR
     (CH >= '0') AND (CH <= '9') OR (CH >= 'a') AND (CH <= 'z')
  END;    { alpha_num }

BEGIN   { main }
  MAKE_NULL (Q);                                      { איפוס התור }
  FOUND_COUNTER := 0;  { איפוס מונה הסופר את מופעי המחרוזת המבוקשת }
  WRITE ('FIND? ');
  READLN (WORD);                                      { קריאת המחרוזת המבוקשת }
  WORD_LENGTH := MAX_WORD_LENGTH;
  WHILE  WORD [WORD_LENGTH] = ' '  DO
    WORD_LENGTH := WORD_LENGTH - 1;                   { מציאת אורך המחרוזת }
  ASSIGN (TEXT_FILE, 'DATA');
  RESET (TEXT_FILE);
  CHAR_COUNTER := 0;
  WORD_LOCATION := 1;
  WHILE  NOT EOF (TEXT_FILE)  DO
   BEGIN
     READ (TEXT_FILE, CH);
     WHILE  NOT EOLN (TEXT_FILE)  AND NOT ALPHA_NUM (CH)  DO
      BEGIN                       { הכנסת לתור של תוים שאינם אלפאנומריים }
        SAVE_IN_Q (Q, CH, CHAR_COUNTER);
        READ (TEXT_FILE, CH)
      END;
     I := 1;
     WHILE  NOT EOLN (TEXT_FILE)  AND  ALPHA_NUM (CH)  DO
      BEGIN          { קריאה של מחרוזת תוים אלפאנומריים ל-TEMP_WORD }
        TEMP_WORD [I] := CH;
        I := I + 1;
        READ (TEXT_FILE, CH)
      END;
     TEMP_WORD [I] := CH;
     IF  EQUAL (WORD, TEMP_WORD, I - 1)  THEN          { השוואה בין המלים }
      BEGIN                                            { יש שיויון }
        PRINT_QUEUE (Q);    { הדפסה מהתור של 20 התוים הקודמים למלה }
        FOUND_COUNTER := FOUND_COUNTER + 1;            { מונה הופעות המלה }
```

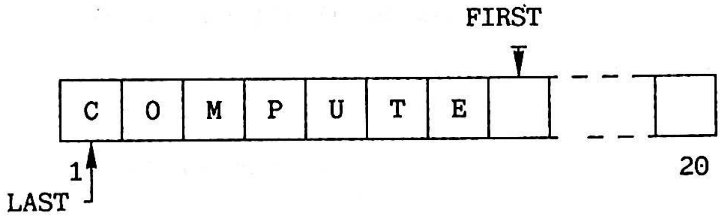
```
        MAKE_NULL (Q);                                        { איפוס התור }
        CHAR_COUNTER := 0;
        FOR  I := 1  TO  WORD_LENGTH + 1  DO
           WRITE (TEMP_WORD [I]);                             { הדפסת המלה המבוקשת }
        I := 1;
        WHILE  NOT EOF (TEXT_FILE) AND (I <= 20)  DO
         BEGIN                                                { הדפסת 20 התווים שלאחר המלה }
           IF  EOLN (TEXT_FILE) THEN
              READLN (TEXT_FILE);
           READ (TEXT_FILE, CH);
           WRITE (CH);
           I := I + 1
         END;   { while }
        WRITELN
      END    { if }
     ELSE                                                     { המלים לא שוות }
        FOR  WORD_LOCATION := 1  TO  I  DO                    { שמירת המלה בתור }
           SAVE_IN_Q (Q, TEMP_WORD [WORD_LOCATION], CHAR_COUNTER);
     IF  EOLN (TEXT_FILE) AND NOT EOF (TEXT_FILE)  THEN
        READLN (TEXT_FILE)                                    { שורה חדשה }
   END;   { while not eof }
  CLOSE (TEXT_FILE);
  WRITELN;
  IF  FOUND_COUNTER = 0  THEN
     WRITELN ('SEARCH STRING NOT FOUND')
  ELSE
     WRITELN ('SEARCH STRING WAS FOUND ', FOUND_COUNTER:1, ' TIMES')
END.
```

דוגמאות להרצת התוכנית:

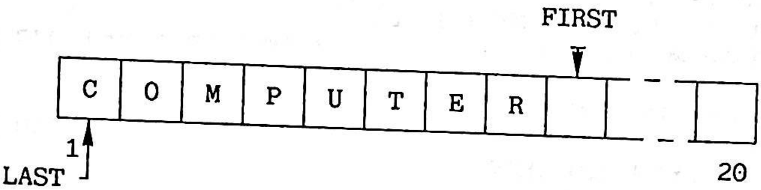
א. תור ריק



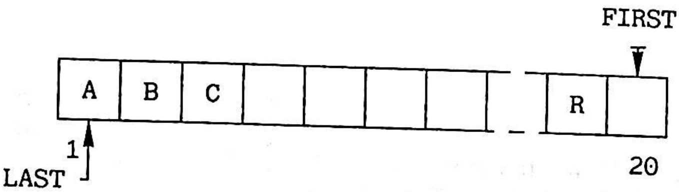
ב. הוספה תו לתור



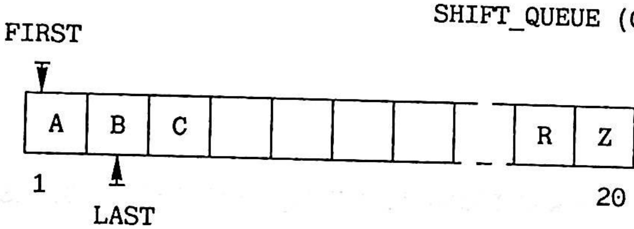
ביצוע הפקודה: ADD_TO_QUEUE (QUEUE, 'R')



ג. הוספה של איבר לתור (מילוי התור)

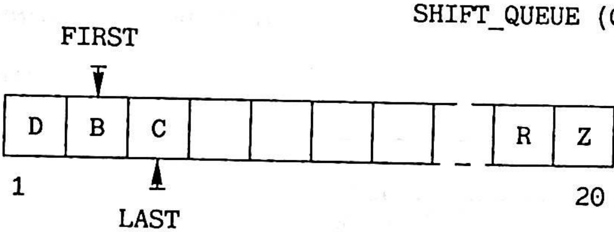


ביצוע הפקודה: SHIFT_QUEUE (QUEUE, 'Z')



ד. הוספה של תו למערך מלא

ביצוע הפקודה: SHIFT_QUEUE (QUEUE, 'D')



```
PROCEDURE BUILD_INDEX_FILE;
   VAR
      F : CHAPTER_FILE;
      C : CHAP_INDEX;
      BOOK_NDX : TEXT;
      TEMP_WORD : WORD_TYPE;
      BUFFER_FULL : BOOLEAN;
      LINE : PACKED ARRAY [1..80] OF CHAR;
    BEGIN
      ASSIGN (F, CHAP_FILE_NAME [1]);
      RESET (F);
      ASSIGN (BOOK_NDX, 'BOOK.NDX');
      REWRITE (BOOK_NDX);
      BUFFER_FULL := FALSE;
      IF  NOT EOF (F)  THEN
       BEGIN
         READ (F, C);
         BUFFER_FULL := TRUE
       END;
      WHILE  NOT EOF (F) OR BUFFER_FULL  DO
       BEGIN
         TEMP_WORD := C.WORD;
         BUFFER_FULL := FALSE;
         WRITE (BOOK_NDX, C.WORD, C.PAGE_NU:4);  { כתיבה בקובץ תמליל }
                                                 { של המלה ומספר הדף הראשון.}
         WHILE  NOT EOF (F) AND (TEMP_WORD = C.WORD)  DO
          BEGIN
            READ (F, C);
            BUFFER_FULL := TRUE;
            IF  TEMP_WORD = C.WORD  THEN
             BEGIN
               BUFFER_FULL := FALSE;
               WRITE (BOOK_NDX, C.PAGE_NU:4)      { כתיבת שאר האיזכורים }
             END                                  {      לקובץ התמליל. }
          END;    { while not eof..and..}
         WRITELN (BOOK_NDX)                        { שורה חדשה בקובץ }
       END;    { while not eof..or..}
      CLOSE (F);
      RESET (BOOK_NDX);
      WRITELN;
      WHILE  NOT EOF (BOOK_NDX)  DO
       BEGIN
         READLN (BOOK_NDX, LINE);  { קריאת שורה מקובץ התמליל והדפסתה }
         WRITE (LINE)
       END;
      CLOSE (BOOK_NDX)
    END;
```

הסבר:
פרוצדורת זו שונה מן הפרוצדורה BUILD_INDEX_FILE שבתוכנית 7 בפרק 11.
מטרת השינוי: שמירה על אינדקס הספר בקובץ תמליל ולא בקובץ המכיל
רשומות. שאר חלקי התוכנית נשארו ללא שינוי.

תוכנית 5

```
PROGRAM CAL_TEXT (TEXT_FILE, OUTPUT);
VAR
   TEXT_FILE : TEXT;
   CH : CHAR;
   LINE_LENGTH,
   AVERAGE,
   LINES_NUMBER,
   MIN,
   MAX   : INTEGER;
BEGIN
   ASSIGN (TEXT_FILE, 'TEXT.DAT');
   RESET (TEXT_FILE);
   LINES_NUMBER := 0;
   AVERAGE := 0;
   MAX := 0;
   MIN := 80;
   WHILE  NOT EOF (TEXT_FILE)  DO                  { לולאה עד סוף הקובץ }
    BEGIN
      LINES_NUMBER := LINES_NUMBER + 1;              { מונה מספר השורות }
      LINE_LENGTH := 0;
      WHILE  NOT EOLN (TEXT_FILE)  DO              { לולאה לקריאת תווים עד }
       BEGIN                                       { סוף שורה בקובץ. }
        READ (TEXT_FILE, CH);
        LINE_LENGTH := LINE_LENGTH + 1               { מונה אורך השורה }
       END;
      IF  NOT EOF (TEXT_FILE)  THEN                 { אם לא סוף הקובץ - }
         READLN (TEXT_FILE);                        { שורה חדשה. }
      IF  LINE_LENGTH > MAX  THEN
         MAX := LINE_LENGTH;                       { עדכון אורך שורה מכסימלי }
      IF  LINE_LENGTH < MIN  THEN
         MIN := LINE_LENGTH;                       { עדכון אורך שורה מינימלי. }
      AVERAGE := AVERAGE + LINE_LENGTH             { סיכום אורכי השורות. }
    END;   { while not eof }
   AVERAGE := ROUND (AVERAGE / LINES_NUMBER);   { ממוצע אורכי השורות }
   WRITELN (' TOTAL ', LINES_NUMBER:1, ' LINES');
   WRITELN ('MAXIMUM LINE LENGTH IS: ', MAX:1);
   WRITELN ('MINIMUM LINE LENGTH IS: ', MIN:1);
   WRITELN ('AVERAGE LINE LENGTH IS: ', AVERAGE:1);
   CLOSE (TEXT_FILE)
END.
```

תוכנית 7

```
PROGRAM FORMATTER (DATA, OUTPUT);
CONST
   MAX_LENGTH = 80;                         { אורך שורה מכסימלי בקובץ התמליל }
TYPE
   LINE_TYPE = PACKED ARRAY [1..MAX_LENGTH] OF CHAR;

VAR
   DATA : TEXT;                             { קובץ תמליל }
   A,                                       { אורך שורה }
   B  : INTEGER;                            { רווח בין השורות }
   CH : CHAR;
   LINE : LINE_TYPE;
   FILE_NAME : PACKED ARRAY [1..20] OF CHAR; { שם פיסי של קובץ תמליל }

   PROCEDURE SPACE (N : INTEGER);           { פרוצדורה להדפסת N רווחים }
      BEGIN
         IF  N > 0  THEN
            WRITE (' ':N)
      END;    { space }

   PROCEDURE LINES (N : INTEGER);           { פרוצדורה להדפסת N שורות רווח }
      BEGIN
         WHILE  N > 0  DO
          BEGIN
            WRITELN;
            N := N - 1
          END
      END;    { lines }

                          { הפרוצדורה LENGHT מחזירה את אורך
                            השורה ללא תווי הרווח שבסופה. }

   FUNCTION LENGTH (LINE : LINE_TYPE) : INTEGER;
      VAR
         I : INTEGER;
      BEGIN
         I := MAX_LENGTH;                                     { אורך מכסימלי }
         WHILE  (LINE [I] = ' ') AND (I > 0)  DO { סריקה מהתו האחרון
            I := I - 1;                            ועד התו הראשון. }
         LENGTH := I
      END;    { length }

                                            { הפרוצדורה CENTER ממרכזת את
   PROCEDURE CENTER (LINE   : LINE_TYPE;      השורה LINE
                     LENGTH : INTEGER);       שאורכה LENGHT. }
      BEGIN
         SPACE ( (A - LENGTH) DIV 2 );      { הדפסת רווחים לפני השורה }
         WRITELN (LINE)                     { הדפסת השורה }
      END;    { center }
```

```
{ הפונקציה מחזירה את התו הבא בקובץ התמליל }
{ או רווח אם זהו סוף שורה או סוף הקובץ.

FUNCTION NEXT_CHAR : CHAR;
   VAR
      CH : CHAR;
   BEGIN
      IF  NOT EOF (DATA) AND NOT EOLN (DATA) THEN
         READ (DATA, CH)
      ELSE
         CH := ' ';
      NEXT_CHAR := CH
   END;     { next_char }

{ הפרוצדורה READ_INPUT קוראת }
{ את השורה הבאה בקובץ התמליל.

PROCEDURE READ_INPUT (VAR LINE : LINE_TYPE);
   VAR
      I : INTEGER;
   BEGIN
      IF  NOT EOF (DATA)  THEN                   { האם זהו סוף הקובץ? }
         READLN (DATA, LINE)                     {        לא - קרא שורה }
      ELSE                                       { כן - זהו סוף הקובץ }
         FOR  I := 1  TO  MAX_LENGTH  DO
            LINE [I] := ' '                 { הפרוצדורה מחזירה שורה ריקה }
   END;     { read_input }

{ הפרוצדורה NEW_INPUT_LINE מקפיצה את המצביע }
{ לקובץ לשורה חדשה (אם לא סוף הקובץ).

PROCEDURE NEW_INPUT_LINE;
   BEGIN
      IF  NOT EOF (DATA)  THEN
         READLN (DATA)
   END;     { new_input }

{ הפונקציה READ_NUMBER קוראת תווי ספרות מקובץ }
{ תמליל עד תו שאינו סיפרה, ומחזירה את ערך המספר.

FUNCTION READ_NUMBER : INTEGER;
   VAR
      CH : CHAR;
      N  : INTEGER;
   BEGIN
      N := 0;
      CH := NEXT_CHAR;
      WHILE  (CH >= '0') AND (CH <= '9')  DO
       BEGIN
         N := 10 * N + ORD (CH) - ORD ('0');
         CH := NEXT_CHAR
```

```
      END;
     NEW_INPUT_LINE;                                          { שורה חדשה }
     READ_NUMBER := N
  END;     { read_number }

 { הפרוצדורה WRITE_LINE קוראת את השורה הבאה מקובץ התמליל ומדפיסה
   אותה לאחר SPC רווחים ולאחר הדפסת התו CH, אם הוא שונה מרווח. }

PROCEDURE WRITE_LINE (SPC : INTEGER;
                      CH : CHAR);
  VAR
     L, I : INTEGER;

  FUNCTION MIN (A, B :INTEGER) : INTEGER;        { מינימום בין A ל-B }
     BEGIN
        IF  A < B  THEN
           MIN := A
        ELSE
           MIN := B
     END;    { min }

  BEGIN    { write_line }
     READ_INPUT (LINE);                          { קריאת שורת קלט הבאה }
     LINES (B);                                  { רווח של B שורות }
     SPACE (SPC);                                { הדפסת SPC רווחים }
     L := A - SPC + 1;              { מספר התווים שנותרו בשורה בשלב זה }
     IF  CH <> '*'  THEN
      BEGIN
        WRITE (CH);
        L := L - 1
      END;
     WHILE  LINE [L] <> ' '  DO                  { לולאה למציאת רווח, }
        L := L - 1;                              { כדי לא לחתוך מלה. }
     FOR  I := 1  TO  L  DO
        WRITE (LINE [I]);                        { הדפסת השורה עד הרווח. }
     WRITELN;                                    { מעבר לשורה חדשה }
     WHILE  L < LENGTH (LINE)  DO       { לולאה כל זמן שנותרו תווים }
      BEGIN                             {  בשורה שלא הודפסו. }
        LINES (B);                               { הדפסת רווח של B שורות }
        I := MIN (L + A, LENGTH (LINE)) + 1;     { מקום החיתוך הבא
                                                   של השורה. }
        WHILE  LINE [I] <> ' '  DO               { לולאה למציאת רווח
           I := I - 1;                            כדי לא לחתוך מלה. }
        WHILE  L < I  DO
         BEGIN
           L := L + 1;
           WRITE (LINE [L])
         END;    { while l<i }
        WRITELN
      END    { while l }
  END;    { write_line }
```

```
BEGIN    { main }
   WRITE ('ENTER FILE NAME: ');
   READLN (FILE_NAME);
   ASSIGN (DATA, FILE_NAME);
   RESET (DATA);
   IF  NOT EOF (DATA)  THEN
      READLN (DATA, A, B);
   WHILE  NOT EOF (DATA)  DO
    BEGIN
      CH := NEXT_CHAR;                                    { תו ראשון בשורה }
      IF  CH = '*'  THEN                   { לאחר ה-'*' תופיע פקודת הדפסה }
       BEGIN
         CASE  NEXT_CHAR  OF
            'S' :  BEGIN                                  { הדפסת שורות רווח }
                     LINES (READ_NUMBER - B); { במקרה זה הודפסו פחות
                        { .WRITE_LINE-שורות רווח, כי הן מודפסות ב B  }
                     WRITE_LINE (0, '*')
                   END;
            'P' :  BEGIN                                       { התחלת פיסקה }
                     LINES (B);
                     WRITE_LINE (READ_NUMBER, '*')
                   END;
            'C' :  BEGIN                                       { מירכוז שורה }
                     NEW_INPUT_LINE;                       { שורה חדשה.    }
                     READ_INPUT (LINE);              {קריאת שורת קלט באה}
                     CENTER (LINE, LENGTH (LINE))               { מירכוז }
                   END
         END      { case }
       END    { if }
      ELSE                                           { התו הראשון אינו '*' }
         WRITE_LINE (0, CH)               { הדפסת שארית השורה אחרי התו CH }
    END   { while not eof }               {         ללא הוספת רווחים לפניה. }
END.
```

פרק 14
קבוצות

שאלות

שאלה 1

א. CH IN ['B'..'L']

ב. [A, B, C] <= [10..15]

<u>הסבר:</u> הקבוצה, שמכילה שלושה משתנים שלמים (A, B ו-C) המייצגים מספרים בין 10 ל-15, היא קבוצה חלקית (שווה או מוכלת) לקבוצת המספרים השלמים בין 10 ל-15.

ג. X IN [20..30] - [23, 29]

שאלה 2

a) [6, 7, 8, 9, 10]

b) FALSE — הסבר: כי [4] <> [ ]

c) [ ] — הסבר: K1 + K5 = K1 ; K2 + K4 = K2 ==>> K1 * K2 = [ ]

d) TRUE — הסבר: K1 - K2 = K1 כי K1 ∩ K2 = ϕ
K1 - K3 = [7, 9]
[7, 9] - K4 = [7, 9]
9 IN [7, 9] = TRUE

f) TRUE — הסבר: [1, 2, 3, 4, 5] <= [1..10]

תוכניות

תוכנית 1

כדי לפתור בעיה זו יש להחליף את שורה 4 בתוכנית PIRUK_MILA שבעמוד 307 שבספר "פסקל" בשורה הרשומה לעיל:

```
(4)        OT_ACHRONA = 'Z';
```

תוכנית 2

לפתרון בעיה זו יש לשנות את התוכנית הרשומה בעמ' 307 שבספר "פסקל". ראה את ההערה בעמ' 308, לאחר התוכנית המקורית.

```
PROGRAM PIRUK_MILA (INPUT, OUTPUT);
CONST
   FIRST = 'A';
   LAST  = 'Z';
   SPACE = ' ';
   WORDS_IN_LINE = 15;
TYPE
   CHARACTERS = FIRST..LAST;
VAR
   WORD_CH,
   TEMP_WORD,
   DOUBLE      : SET OF CHARACTERS;
   CH1,
   CH2,
   CH3         : CHARACTERS;
   WORD : ARRAY [1..20] OF CHARACTERS;
   I    : INTEGER;
BEGIN
   READLN (WORD);
   WORD_CH := [];                        { קבוצת האותיות המופיעה במלה }
   DOUBLE := [];                         { קבוצת האותיות הכפולות במלה }
   I := 1;
   WHILE  WORD [I] IN ['A'..'Z']  DO
    BEGIN          { טיפול באותיות עד שמגיעים לאות שאינה בין 'A' ל-'Z'. }
      IF  WORD [I] IN WORD_CH  THEN        { אם האות כבר נמצאת בקבוצת }
         DOUBLE := DOUBLE + [WORD [I]]     { האותיות המופיעות במלה, }
                          { מוסיפים אות זו לקבוצת האותיות הכפולות, }
      ELSE                           { אחרת מוסיפים אותה לקבוצת האותיות. }
         WORD_CH := WORD_CH + [WORD [I]];
      I := I + 1
   END;
```

```
    I := 0;
    FOR  CH1 := FIRST  TO  LAST  DO
       IF  CH1 IN WORD_CH  THEN
        BEGIN                                          { אם האות הראשונה       }
          TEMP_WORD := (WORD_CH - [CH1]) + DOUBLE;     { נמצאת בקבוצת האותיות. }
                                                       { איחוד של קבוצת        }
                                                         האותיות הכפולות עם קבוצת }
                                                       { אותיות המלה, פחות האות שנבחרה. }
          FOR  CH2 := FIRST  TO  LAST  DO
             IF  CH2 IN TEMP_WORD  THEN
              BEGIN
                TEMP_WORD := (TEMP_WORD - [CH2]) + DOUBLE;
                FOR  CH3 := FIRST  TO  LAST  DO
                   IF  (CH3 IN TEMP_WORD) AND
                       ([CH3] <> [CH1, CH2])  THEN
                    BEGIN          { האות השלישית שייכת לאותיות המלה }
                                   { ואין שיוויון (בזוגות) בין שלוש האותיות. }
                      WRITE (CH1, CH2, CH3, SPACE);      { הדפסת המלה }
                      I := I + 1;
                      IF  I = WORDS_IN_LINE  THEN
                       BEGIN
                         I := 0;
                         WRITELN                         { שורה חדשה }
                       END
                    END;   { if }
                TEMP_WORD := TEMP_WORD + [CH2]   { החזרת האות השניה }
                                                 { לקבוצה, כדי שהיא תוכל }
                                                 { להבחר כ-CH3 במעבר הבא. }
              END    { if ch2 }
        END    { if ch1 }
END.
```

תוכנית 3

```
PROGRAM SETS (OUTPUT);
CONST
   FIRST = 1;
   LAST  = 50;
TYPE
   SUG_BASIS    = FIRST..LAST;
   SUG_KEVUTZA = SET OF SUG_BASIS;
VAR
   K1,
   K2,
   K3 : SUG_KEVUTZA;
   I  : SUG_BASIS;
   RISHON : BOOLEAN;
```

```
BEGIN
   WRITE ('[');
   RISHON := TRUE;
   FOR  I := FIRST  TO  LAST  DO
{A}    IF  I IN ((K1 + K2) - K1 * K2)  THEN          { לפתרון חלק א' }
        BEGIN                { ((K1UK2)  - (K1nK2))  :הביטוי הלוגי הוא }
          IF  NOT RISHON  THEN
             WRITE (', ')
          ELSE
             RISHON := FALSE;
          WRITE (I:1)
        END;
   WRITELN (']')
END.
```

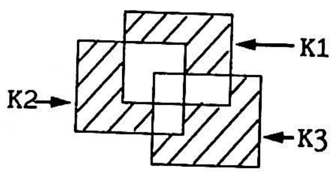
<u>הערה:</u>
הפקודה IF בשורה המסומנת ב-A מתאימה לפתרון חלק א' של השאלה. כדי לפתור את שאר סעיפי השאלה, יש להחליף פקודה זו בפקודות אחרות, כפי שיוסבר בהמשך.
לפתרון שאר הסעיפים צרפנו הסבר, אשר כולל את הביטוי הלוגי של הפתרון ותיאור גרפי שלו. החלק המקווקו בשרטוטים מייצג את הפתרון.

n - חיתוך ; U - איחוד

<u>חלק ב'</u> - הפקודה שיש לרשום בשורה A בתוכנית:

```
{B}   IF  I IN ((K1 + K2 + K3) - K1 * K2 - K2 * K3 - K1 * K3)  THEN
```

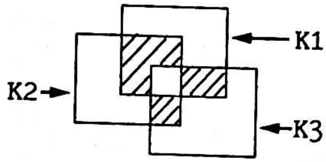
הסבר: (K1 U K2 U K3) - (K1 n K2) - (K2 n K3) - (K1 n K3)



<u>חלק ג'</u> - הפקודה שיש לרשום בשורה A בתוכנית:

```
{C}   IF  I IN ((K1 * K2 + K2 * K3 + K1 * K3) - K1 * K2 * K3)  THEN
```

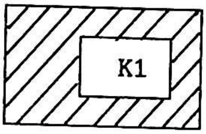
הסבר: $(K1 \cap K2) \cup (K2 \cap K3) \cup (K1 \cap K3) - (K1 \cap K2 \cap K3)$



<u>חלק ד'</u> - הפקודה שיש לרשום בשורה A בתוכנית:

```
{D}  IF  NOT (I IN K1)  THEN
```

הסבר: $\overline{K1}$



<u>הערה:</u> הקו שמעל הביטוי מסמן "משלים".

<u>חלק ה'</u> - הפקודה להחלפה:

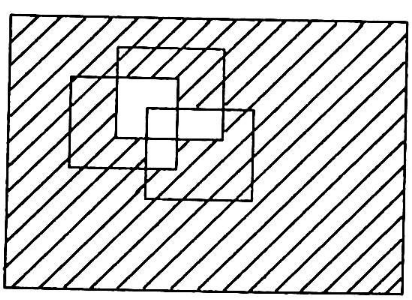
```
{E}   IF  NOT (I IN K1 + K2 + K3)  THEN
```

הסבר: $\overline{K1 \cup K2 \cup K3}$

<u>חלק ו'</u> - הפקודה להחלפה:

```
{F}   IF  NOT (I IN  K1 * K2 + K2 * K3 + K1 * K3)   THEN
```

הסבר: $\left[\overline{(K1 \cap K2) \cup (K2 \cap K3) \cup (K1 \cap K3)}\right]$



תוכנית 4

```
PROGRAM COUNT_CHAR (TEXTFILE, OUTPUT);
TYPE
   CHARACTERS = SET OF CHAR;
VAR
   TEXTFILE : TEXT;
   CH       : CHAR;
   VOWELS_COUNTER,
   CHAR_COUNTER   : INTEGER;
   VOWELS,
   OTHERS         : CHARACTERS;
BEGIN
   VOWELS := ['A', 'E', 'I', 'O', 'U'];          { אותיות שהן תנועות }
   OTHERS := ['A'..'Z'] - VOWELS;                 { שאר האותיות }
   VOWELS_COUNTER := 0;                           { מונה תנועות }
   CHAR_COUNTER := 0;                             { מונה שאר האותיות }
   ASSIGN (TEXTFILE, 'TEXT.DAT');                 { פתיחת קובץ TEXT }
   RESET (TEXTFILE);
   WHILE  NOT EOF (TEXTFILE)  DO
    BEGIN
      READ (TEXTFILE, CH);
      IF  CH IN VOWELS  THEN        { אם האות שייכת לקבוצת התנועות }
         VOWELS_COUNTER := VOWELS_COUNTER + 1        { הוסף אחד }
                                                     { למונה התנועות }
      ELSE
         IF  CH IN OTHERS  THEN  { אם האות שייכת לקבוצת שאר האותיות }
            CHAR_COUNTER := CHAR_COUNTER + 1;        { הוסף אחד }
                                                { למונה שאר האותיות }
      IF  EOLN (TEXTFILE) AND NOT EOF (TEXTFILE)  THEN
         READLN (TEXTFILE)                           { סוף שורה בקובץ }
    END;   { while }
   WRITELN (VOWELS_COUNTER, ' - מספר התנועות שווה');
   WRITELN (CHAR_COUNTER, ' - מספר שאר האותיות שווה');
   CLOSE (TEXTFILE)
END.
```

תוכנית 5

```
PROGRAM COUNT_CHAR (TEXTFILE, OUTPUT);
TYPE
   CHARACTERS = SET OF CHAR;
VAR
   TEXTFILE : TEXT;
   CH1, CH2 : CHAR;
   VOWELS_COUNTER : INTEGER;
   VOWELS : CHARACTERS;
```

```
BEGIN
   VOWELS := ['A', 'E', 'I', 'O', 'U'];                    { קבוצת התנועות }
   VOWELS_COUNTER := 0;                                     { מונה הסדרות }
   ASSIGN (TEXTFILE, 'TEXT.DAT');                           { פתיחת קובץ TEXT }
   RESET (TEXTFILE);
   WHILE  NOT EOF (TEXTFILE)  DO
    BEGIN
      READ (TEXTFILE, CH1);                                 { קריאת תו ראשון }
      WHILE  NOT EOLN (TEXTFILE)  DO
       BEGIN
         READ (TEXTFILE, CH2);                              { קריאת תו שני }
         IF  [CH1, CH2] <= VOWELS  THEN
          BEGIN          { קבוצת שני התווים הינה חלקית לקבוצת התנועות }
            VOWELS_COUNTER := VOWELS_COUNTER + 1;    { הוספת 1 למונה
                                                       הסדרות. }
            WRITELN (CH1, CH2)                              { הדפסת הסדרה }
          END;
         CH1 := CH2 { העברת האות השניה למשתנה השומר את האות הראשונה }
       END
    END;   { while not eof }
   WRITELN (VOWELS_COUNTER, ' - מספר הסדרות שווה');
   CLOSE (TEXTFILE)
END.
```

תוכנית 6

```
PROGRAM COUNT_CHAR (TEXTFILE, OUTPUT);
TYPE
   CHARACTERS = SET OF CHAR;
VAR
   TEXTFILE     : TEXT;
   CH           : CHAR;
   CHAR_COUNTER,
   MAX_CHAR,
   I, J         : INTEGER;
   WORD_CHARS,
   LETTERS      : CHARACTERS;
   WORD,
   MAX_WORD     : ARRAY [1..20] OF CHAR;
BEGIN
   MAX_CHAR := 0;
   LETTERS := ['A'..'Z'];
   ASSIGN (TEXTFILE, 'TEXT.DAT');
   RESET (TEXTFILE);
```

```
    WHILE  NOT EOF (TEXTFILE)  DO
     BEGIN
       I := 1;
       WORD_CHARS := [];                              { קבוצת האותיות במלה }
       CHAR_COUNTER := 0;                    { מונה של האותיות השונות במלה }
       READ (TEXTFILE, CH);
       WHILE  NOT EOLN (TEXTFILE) AND (CH IN LETTERS) DO { לולאה, כל }
        BEGIN                                 { זמן שלא סוף שורה והתו הוא אות }
          WORD [I] := CH;
          I := I + 1;
          IF  NOT (CH IN WORD_CHARS)  THEN
           BEGIN                          { נקראה אות שעדיין אינה מופיעה במלה }
             WORD_CHARS := WORD_CHARS + [CH];     { הוספת האות לקבוצת }
                                                  {   האותיות במלה.   }
             CHAR_COUNTER := CHAR_COUNTER + 1       { הוספת אחד למונה }
                                                    {  האותיות השונות.}
           END;
          READ (TEXTFILE, CH)                           { קריאת התו הבא }
        END;   { while not eoln }
       WORD [I] := ' ';                                  { סימון סוף מלה }
       IF  CHAR_COUNTER > MAX_CHAR  THEN
        BEGIN                                 { אם במלה יש יותר אותיות שונות }
          MAX_WORD := WORD;              { עדכון המשתנה של המלה הארוכה ביותר }
          MAX_CHAR := CHAR_COUNTER       { עדכון מספר האותיות המכסימלי }
        END;
       IF  EOLN (TEXTFILE) AND NOT EOF (TEXTFILE) THEN
          READLN (TEXTFILE)              {.מצב של סוף שורה אך לא של סוף קובץ }
                                                   { יש לקרוא שורה חדשה. }
     END;   { while not eof }
    I := 1;
    WHILE  MAX_WORD [I] IN LETTERS  DO
     BEGIN                                                  { הדפסת המלה }
       WRITE (MAX_WORD [I]);
       I := I + 1
     END;
    WRITELN ( ' במלה ', MAX_CHAR:2,
              ' מספר האותיות השונות הגדול ביותר הוא');
    CLOSE (TEXTFILE)
END.
```

פרק 15
רקורסיה

שאלות

שאלה 1

התוכנית מדפיסה משולש של מספרים, לדוגמה:

כאשר SIZE = 5

```
1
121
12321
1234321
123454321
```

מספר השורות בפלט יהיה כגודלו של SIZE.

שאלה 3

לפתרון שאלה זו יש להוסיף בתוכנית HIPUCH_KELET (עמ' 327 בספר "פסקל") את המלה BEGIN לפני שורה 7, ו-END אחרי שורה 8.

תוכניות

תוכנית 1

```
PROGRAM READ_CH (INPUT, OUTPUT);

PROCEDURE READ_PRINT_CHAR;
   VAR
      CH : CHAR;
   BEGIN
      READ (CH);                  { קרא תו }
      IF  CH <> ' '  THEN         { אם התו שונה מרווח }
       BEGIN
         WRITE (CH);              { הדפס תו }
         READ_PRINT_CHAR          { קרא רקורסיבית לשאר התווים }
       END
   END;

BEGIN  { main }
   READ_PRINT_CHAR
END.
```

תוכנית 2 א'

פתרון בשיטת הרקורסיה:

```
FUNCTION RECURSIVE_FACT (N : INTEGER) : REAL;
   BEGIN
      IF  N = 0  THEN                                { אם N = 0 }
         RECURSIVE_FACT := 1                         { אז N!=1 }
      ELSE                                           { אחרת: N! = N * (N - 1)! }
         RECURSIVE_FACT := N * RECURSIVE_FACT (N - 1)
   END;
```

פתרון איטרטיבי:

```
FUNCTION ITERATIVE_FACT (N : INTEGER) : REAL;
   VAR
      F : REAL;
   BEGIN
      F := 1;
      WHILE  N > 0  DO
       BEGIN
         F := F * N;                          { F=N*(N-1)*(N-2)*...*1 }
         N := N - 1
       END;
      ITERATIVE_FACT := F
   END;
```

תוכנית 2 ב'

פתרון בשיטת הרקורסיה:

```
FUNCTION MULT (N, M :INTEGER) : INTEGER;
BEGIN
   IF  M = 1  THEN                                        { אם N = 1 }
      MULT := N                              { if M=1 then N*1=N  אז }
   ELSE                                                      { אחרת }
      MULT := N + MULT (N, M - 1)           { N*M = N + N * (M - 1) }
END;
```

פתרון איטרטיבי:

```
FUNCTION MULT (N, M :INTEGER) : INTEGER;
VAR
   I, SUM : INTEGER;
BEGIN
   SUM := 0;
   FOR  I := 1  TO  M  DO
      SUM := SUM + N;                      { N*M = N + N + N +...+ N }
   MULT := SUM                             {           └──── m-1 ──┘ }
END;                                       {              פעמים      }
```

תוכנית 2 ג'

פתרון בשיטת הרקורסיה:

```
FUNCTION FIB (N :INTEGER) : INTEGER;
BEGIN
   IF  N IN [0, 1]  THEN
      FIB := N              { if N=0 then FIB=0 ; if N=1 then FIB=1 }
   ELSE
      FIB := FIB (N - 1) + FIB (N - 2)
END;
```

הסבר:

פונקציה זו אינה פונקציה יעילה.

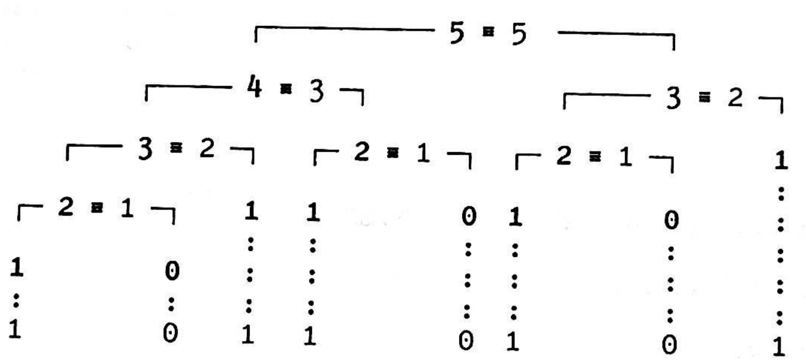
לדוגמה:

```
FIB (3) = ?
FIB (3) = FIB (2) + FIB (1) = 1 + 1 =  2

FIB (2) = FIB (1) + FIB (0) = 1 + 0 = 1
FIB (1) = 1
```

נסה להריץ את הרקורסיה הזו עם ערכים גדולים יותר, והשווה את זמן הביצוע לעומת הרצה של הפתרון האיטרטיבי.

נבחן מה קורה כאשר נרצה לחשב FIB (5), נציג את תהליך החישוב:



הסבר לתרשים:
המספרים המודגשים הם הערכים שנשלחו כפרמרים בכל פעם שבוצעה קריאה לפרוצדורה FIB.
הערכים בשורה התחתונה הם הערכים הסופיים של הפונקציה. עם ערכים אלה תתבצע החזרה לנקודת המוצא.

2 ■ 3 פירושו: בוצעה הפקודה FIB(3) והערך שהחזירה הפונקציה הוא 2.

לפתרון FIB(5) בשיטה הרקורסיבית, יתבצעו 15! קריאות לפונקציה.

פתרון איטרטיבי:

```
FUNCTION FIB (N :INTEGER) : INTEGER;
VAR
   I, SUM, F1, F2 : INTEGER;
BEGIN
   SUM := 0;                        { איפוס הסכום }
   F1  := 0;                        { האיבר הראשון שווה אפס }
   F2  := 1;                        { האיבר השני שווה לאחד }
   FOR  I := 1  TO  N  DO
    BEGIN
      SUM := F1 + F2;               { חישוב האיבר הבא }
      F1  := F2;                    { עדכון שני האיברים האחרונים בסדרה }
      F2  := SUM
    END;
   FIB := SUM
END;
```

תוכנית 3

```
(1)  FUNCTION MISPAR (VAR POWER : INTEGER) : INTEGER;
(2)  VAR
(3)     CH : CHAR;
(4)     I  : INTEGER;
(5)  BEGIN
(6)     READ (CH);
(7)     IF  CH IN ['0'..'9']  THEN
(8)         BEGIN
(9)           I := MISPAR (POWER);
(10)          MISPAR :=  I + POWER * (ORD (CH) - ORD ('0'));
(11)          POWER := POWER * 10
(12)        END
(13)    ELSE
(14)        BEGIN
(15)          MISPAR := 0;
(16)          POWER  := 1
(17)        END
(18) END;
```

<u>הסבר:</u>

תחילתה של הפונקציה הרקורסיבית הוא בקריאת התווים בזה אחר זה (שורה 6). כל עוד התו הנקרא נמצא בתחום '0'..'9', הפונקציה מופעלת בתהליך רקורסיבי (שורה 9). התהליך מגיע לסופו כאשר נקרא תו שאיננו שייך לתחום '0'..'9'. בשלב זה הפונקציה MISPAR מחזירה ערך 0 ומשתנה POWER מקבל ערך 1 והתהליך הרקורסיבי מתחיל בדרכו חזרה.
משמעות החזרת הערך 0 בשורה 15 היא, ש-I קיבל את הערך 0. לכן בחישוב של שורה 10 ישנם הערכים הבאים: I:=0 ; POWER:=1 ו-CH הוא הספרה האחרונה,

(בדוגמה CH3). המשתנה POWER מוכפל ב-10.
בשלב הבא המשתנה I מקבל את הערך שחושב לעיל. המשך ההסבר ראה לפי הדוגמה: השימוש במשתנה I מיועד לקבוע בודאות את סדר החישוב, כי בשלב הקריאה למשתנה POWER עדיין אין לו ערך מתאים. בקריאה לפונקציה יש לשים לב לכך שחייבים להעביר משתנה מסוג INTEGER ואין צורך לתת לו ערך משמעותי כלשהו, כי הוא משמש עזר לחישוב בלבד.

דוגמה:

```
CH_1='5'
CH_2='2'
CH_3='3'
CH_4='^' MISPAR := 0;  POWER := 1;
MISPAR_3 =    0      + POWER(=1) * (ORD (CH_3) - ORD ('0');  POWER = 10
MISPAR_2 = MISPAR_3 + 10         * (ORD (CH_2) - ORD ('0');  POWER = 100
MISPAR_1 = MISPAR_2 + 100        * (ORD (CH_1) - ORD ('0');
```

תוכנית 4

```
FUNCTION OCTAL (DECIMAL : INTEGER) : INTEGER;
VAR
   I : INTEGER;
BEGIN
   IF  DECIMAL > 0  THEN
      OCTAL :=  DECIMAL MOD 8 + 10 * OCTAL (DECIMAL DIV 8)
   ELSE
      OCTAL := 0
END;
```

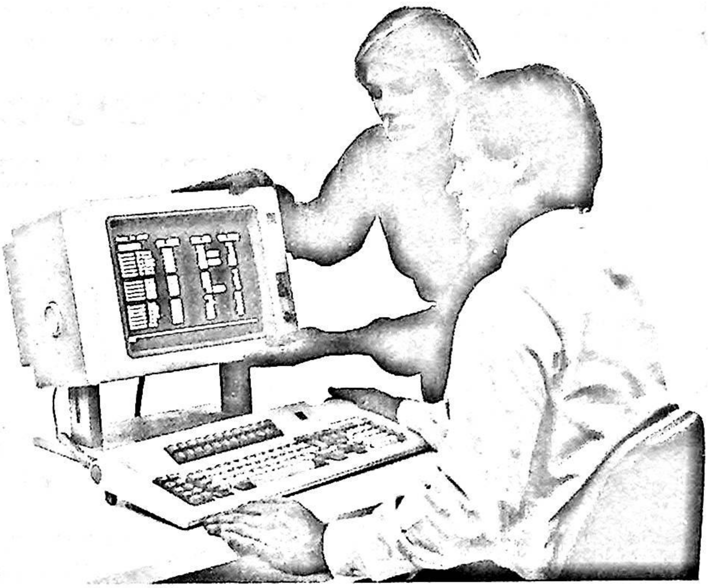
לדוגמה: OCTAL(14)

```
OCTAL_1 := 14 MOD 8 + 10 * OCTAL (14 DIV 8) =  6 + 10 * 1 = 16

OCTAL_2 := 1 MOD 8 + 10 * OCTAL (1 DIV 8) = 1 + 0 = 1

OCTAL_3 := 0
```

```
FUNCTION GCF (N, M :INTEGER) : INTEGER;
BEGIN
   IF  M = N  THEN
      GCF := N
   ELSE
      IF  M < N  THEN
         GCF := GCF (M, N - M)
      ELSE                                          { M > N }
         GCF := GCF (N, M - N)
END;
```



פרק 16
מבני נתונים
ניהול מבני נתונים

<u>כללי</u>

את התרגילים שבפרק זה, מקובל לפתור על ידי מבני נתונים דינמיים, אשר נלמדים בפרק 17 בספר "פסקל". מכיון שעדיין אין אנו מכירים מבנה נתונים זה, יש לפתור את התרגילים ע"י שימוש במערכים. נעשה זאת באמצעות מערכת של ניהול זיכרון, שתפעל ב"רקע". המערכת תחזיק בכל שלב <u>**רשימה של המקומות ה"פנויים"**</u> שאפשר להשתמש בהם. הקצאה או פינוי של מקום (החזרתו לרשימה) יבוצעו ע"י קריאה לפרוצדורות של המערכת.

<u>שים לב:</u> הפרוצדורות המצורפות מנהלות אך ורק את <u>**המקומות הפנויים**</u> בזיכרון ואינן מטפלות ביישום של השימוש במקום הפנוי. תפקיד זה מוטל על תוכנית היישום.

<u>לדוגמה:</u> הפרוצדורה OUR_NEW יודעת להקצות מקום פנוי. אם היישום הוא ניהול של רשימה משורשרת או תור, התוכנית שקיבלה את המקום הפנוי, צריכה לדאוג לשבץ אותו במקום המתאים ברשימה או בתור, לפי העניין.

להלן ניתנות מספר פרוצדורות שרות המדמות מבני נתונים דינמיים בעזרת רשומות ומערכים. בתרגיל התכנות מס' 1 שבהמשך, השתמשנו בפרוצדורות אלו לאחר שהתאמנו אותן לבעיה שהיה עלינו לפתור.

```
CONST
  MEM_SIZE = 50;                              { מספר מכסימלי של רשומות }
  NULL = 0;                                   { סימון סוף רשימה }

TYPE
   POINTER = INTEGER;                         { המצביע הוא אינדקס למקום במערך }
   RECORD_TYPE = RECORD                       { רשומה }
                    ELEMENT : ELEMENT_TYPE;   { איבר מסוג כלשהו }
                    NEXT    : POINTER         { מצביע לרשומה הבאה }
                 END;

   MEMORY = RECORD                            { רשימת מקומות פנויים בזיכרון }
              FIRST_FREE : POINTER;           { מצביע לראש רשימת
                                                המקומות הפנויים. }
              SPACE      : ARRAY [1..MEM_SIZE] OF RECORD_TYPE
            END;                              { מרחב הזיכרון }

VAR
   MEM : MEMORY;

      { פרוצדורה ל"איפוס הזיכרון". היא יוצרת רשימה מקושרת של מקומות
        פנויים מכל הרשומות במערך SPACE. FIRST_FREE מצביע לראש הרשימה. }

   PROCEDURE INITIAL_MEM (VAR M : MEMORY);
      VAR
         PTR : POINTER;
      BEGIN
        WITH  M  DO
         BEGIN
          FOR  PTR := 1  TO  MEM_SIZE - 1  DO
             SPACE [PTR].NEXT := PTR + 1;
          SPACE [MEM_SIZE].NEXT := NULL; { סוף רשימת המקומות הפנויים }
          FIRST_FREE := 1      { מצביע לראש הרשימה, המקום הראשון במערך }
         END
      END;     { initial_mem }

                                      { פרוצדורה להקצאת מקום פנוי.
                                        המצביע למקום הפנוי מוחזר במשתנה PTR. }

   PROCEDURE OUR_NEW (VAR M   : MEMORY;
                      VAR PTR : POINTER);
   BEGIN
       WITH  M  DO
          IF  FIRST_FREE <> NULL  THEN
           BEGIN                  { עדיין לא סוף רשימת המקומות הפנויים. }
             PTR := FIRST_FREE;   { הקצאת המקום הראשון הפנוי ברשימה. }
             FIRST_FREE := SPACE [FIRST_FREE].NEXT  { עדכון המצביע
           END                                 לראש רשימת המקומות הפנויים. }
          ELSE                                 { סוף רשימת המקומות הפנויים }
             WRITELN ('ERROR! - MEMORY FULL')
      END;     { our_new }
```

{ המקום במערך חוזר לרשימת המקומות הפנויים.
שיחרור הרשומה ש-PTR מצביע עליה. }

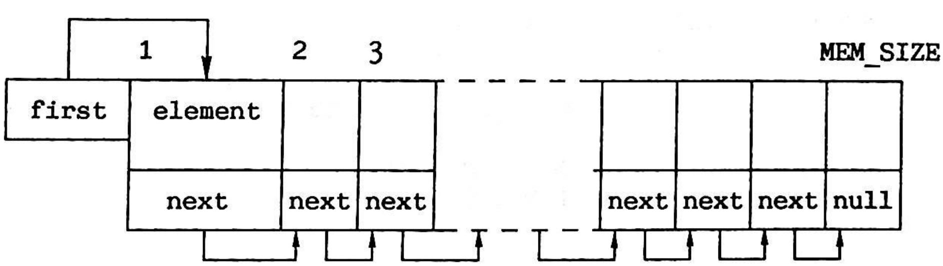
```
PROCEDURE OUR_DISPOSE (VAR M   : MEMORY;
                           PTR : POINTER);
  BEGIN
    IF  PTR  IN [1..MEM_SIZE]  THEN    { בדיקה אם המצביע ב"תחום" }
      WITH  M  DO
       BEGIN                           { שירשור הרשומה לראש רשימת
         SPACE [PTR].NEXT := FIRST_FREE;   המקומות הפנויים. }
         FIRST_FREE := PTR             { עדכון המצביע לראש הרשימה }
       END
    ELSE                               { המצביע אינו ב"תחום" הזיכרון }
      WRITELN ('ERROR! - ILLEGAL POINTER')
  END;    { our_dispose }
```

{ הפונקציה NEXT מחזירה מצביע לרשומה
שנמצאת אחרי זו ש-PTR מצביע עליה. }

```
FUNCTION NEXT (VAR M   : MEMORY;
                   PTR : POINTER) : POINTER;
  BEGIN
    IF  PTR  IN [1..MEM_SIZE]  THEN    { בדיקה אם המצביע ב"תחום" }
      WITH  M  DO
           NEXT := SPACE [PTR].NEXT      { מצביע לרשומה שאחרי }
    ELSE                             { המצביע אינו ב"תחום" הזיכרון. }
      WRITELN ('ERROR! - ILLEGAL POINTER')
  END;    { next }
```

נציג את ניהול הזיכרון באופן סכימטי:

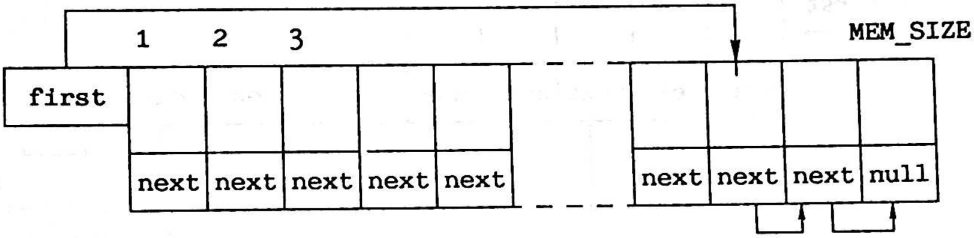
א. רשומת MEMORY לאחר ביצוע הפרוצדורה INITIAL_MEM.

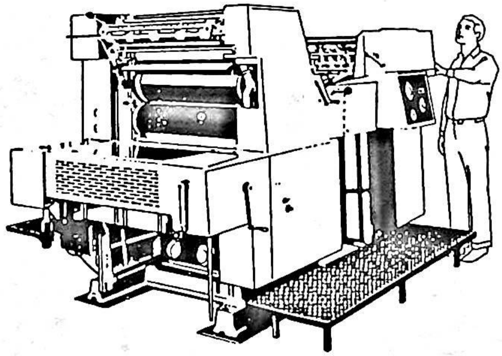


המשתנה first מצביע על הרשומה הראשונה בשרשרת המקומות הפנויים, אשר מכילה את כל הרשומות במערך.

ב. מצב ה"זיכרון" לאחר ביצוע OUR_NEW.

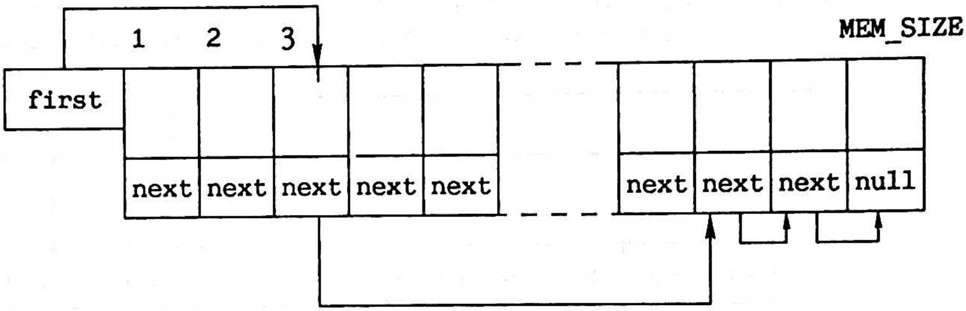
לאחר מספר הקצאות "זיכרון" (ע"י OUR_NEW), נוכל להבחין שיש ב"זיכרון" איברים תפוסים ורשימה מקושרת של המקומות שנותרו פנויים. השימוש במקומות הפנויים יהיה לפי דרישות היישום.



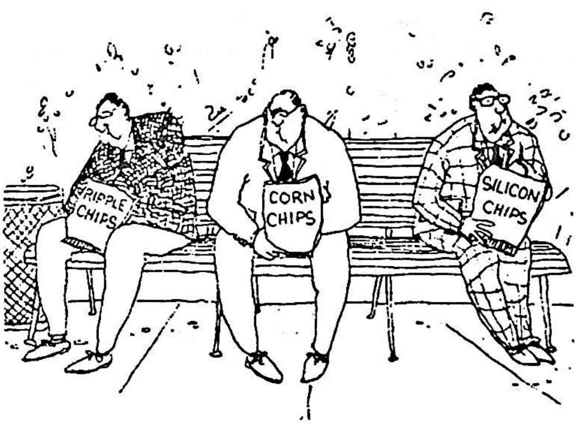


הדפסה

ג. מצב הזיכרון לאחר ביצוע הפקודה OUR_DISPOSE.
לאחר הפקודה OUR_DISPOSE(mem, 3) נקבל:



תוכל להבחין שהפקודה OUR_DISPOSE(mem, 3) שחררה את המקום השלישי במערך והוסיפה אותו אל ראש רשימת המקומות ה"פנויים". פרוצדורה זו אינה מנהלת את הרשימה המשורשת, שאליה היה שייך האיבר במקום קודם לכן. ניהול הרשימה של המקומות הלא-פנויים צריך להתבצע בפרוצדורה הקוראת ל-OUR_DISPOSE. דוגמה בפרוצדורה DELETE בתוכנית שלהלן.



תוכניות

תוכנית 1

```
PROGRAM STD_LIST (INPUT, OUTPUT);
CONST
   NAME_LENGTH = 30;
   MEM_SIZE = 50;
   NULL = 0;

TYPE
   NAME_TYPE = PACKED ARRAY [1..NAME_LENGTH] OF CHAR;
   MARK = 0..100;
   POINTER = INTEGER;
   STD_PTR = POINTER;

   STUDENT_DETAILS = RECORD                          { רשומת פרטי סטודנט }
                        NAME         : NAME_TYPE;
                        EXERCISE     : ARRAY [1..10] OF CHAR;
                        MID_TEST,
                        FINAL_TEST,
                        AVERAGE      : MARK;
                        NEXT         : STD_PTR
                     END;
   STD_FILE = FILE OF STUDENT_DETAILS;

   MEMORY = RECORD                             { רשומה המייצגת את ה"זיכרון" }
               FIRST_FREE : POINTER;
               SPACE : ARRAY [1..MEM_SIZE] OF STUDENT_DETAILS
            END;

VAR
   MEM : MEMORY;
   CLASS    : STD_FILE;
   LIST,
   PTR      : STD_PTR;
   COMMAND  : CHAR;
   STD_NAME : NAME_TYPE;
   AVERAGE_LEVEL : MARK;
```

```
                    { פרוצדורה ל"איפוס הזיכרון". היא יוצרת רשימה מקושרת
                          של מקומות פנויים מכל הרשומות במערך SPACE.
                                  FIRST_FREE } מצביע לראש הרשימה.

PROCEDURE INITIAL_MEM (VAR M : MEMORY);
   VAR
     PTR : POINTER;
   BEGIN
     WITH  M  DO
      BEGIN
       FOR  PTR := 1  TO  MEM_SIZE - 1  DO
           SPACE [PTR].NEXT := PTR + 1;
       SPACE [MEM_SIZE].NEXT := NULL; { סוף רשימת המקומות הפנויים }
       FIRST_FREE := 1
      END
   END; { initial_mem }

                                { פרוצדורה להקצאת מקום פנוי.
                                 המצביע למקום הפנוי מוחזר במשתנה PTR. }

PROCEDURE OUR_NEW (VAR M   : MEMORY;
                   VAR PTR : POINTER);
   BEGIN
      WITH  M  DO
         IF  FIRST_FREE <> NULL  THEN
          BEGIN
            PTR := FIRST_FREE;
            FIRST_FREE := SPACE [FIRST_FREE].NEXT
          END
         ELSE
            WRITELN ('ERROR! - MEMORY FULL')
   END; { our_new }

              { הפרוצדורה OUR_DISPOSE משחררת רשומה ש-PTR מצביע עליה.
                              המקום במערך חוזר לרשימת המקומות הפנויים. }

PROCEDURE OUR_DISPOSE (VAR M   : MEMORY;
                           PTR : POINTER);
   BEGIN
      IF  PTR  IN [1..MEM_SIZE]  THEN
         WITH  M  DO
          BEGIN
            SPACE [PTR].NEXT := FIRST_FREE;
            FIRST_FREE := PTR
          END
      ELSE
         WRITELN ('ERROR! - ILLEGAL POINTER')
   END;    { our_dispose }
```

{ הפונקציה NEXT מחזירה מצביע לרשומה
שאחרי הרשומה ש-PTR מצביע עליה. }

```
FUNCTION NEXT (VAR M   : MEMORY;
                   PTR : POINTER) : POINTER;
   BEGIN
      IF  PTR  IN [1..MEM_SIZE]  THEN
         WITH  M  DO
               NEXT := SPACE [PTR].NEXT
      ELSE
         WRITELN ('ERROR! - ILLEGAL POINTER')
   END;     { next }
```

{ הפונקציה END_OF_LIST מחזירה ערך TRUE כאשר
המצביע LIST_PTR הוא NULL, כלומר - סוף הרשימה. }

```
FUNCTION END_OF_LIST (LIST_PTR : STD_PTR) : BOOLEAN;
   BEGIN
      END_OF_LIST := LIST_PTR = NULL
   END;     { end_of_list }
```

{ הפונקציה LOAD_FROM_FILE יוצרת רשימה מקושרת
מהרשומות הנמצאות בקובץ CLASS ומחזירה מצביע לראש הרשימה. }

```
FUNCTION LOAD_LIST_FROM_FILE (VAR CLASS : STD_FILE) : STD_PTR;
```

{ הפרוצדורה READ_FILE קוראת רשומה אחת מקובץ CLASS,
לתוך רשימה מקושרת, ומחזירה מצביע לאותה רשומה. }

```
   FUNCTION READ_FILE : STD_PTR;
      VAR
         TEMP_PTR : STD_PTR;
      BEGIN
         IF  EOF (CLASS)  THEN
            READ_FILE := NULL
         ELSE
          BEGIN
            OUR_NEW (MEM, TEMP_PTR);
            READ (CLASS, MEM.SPACE [TEMP_PTR]);
            MEM.SPACE [TEMP_PTR].NEXT := READ_FILE;
            READ_FILE := TEMP_PTR
          END
      END;     { read_file }

   BEGIN    { load_list_from_file }
      RESET (CLASS);
      LOAD_LIST_FROM_FILE := READ_FILE
   END;     { load_list_from_file }
```

{ הפרוצדורה SAVE_LIST_IN_FILE כותבת לקובץ CLASS את הרשומות,
שנמצאות ברשימה מקושרת ש-LIST_PTR מצביע עליה. }

```
PROCEDURE SAVE_LIST_IN_FILE (LIST_PTR : STD_PTR;
                         VAR CLASS    : STD_FILE);
  BEGIN
     REWRITE (CLASS);
     WHILE  NOT END_OF_LIST (LIST_PTR)  DO
      BEGIN
        WRITE (CLASS, MEM.SPACE [LIST_PTR]);
        LIST_PTR := NEXT (MEM, LIST_PTR)
      END;
     CLOSE (CLASS)
  END;      { save_list_in_file }
```

{ הפרוצדורה READ_NAME קוראת מחרוזת של תווים
לתוך מערך NAME. סוף המחרוזת מסומן בנקודה. }

```
PROCEDURE READ_NAME (VAR NAME : NAME_TYPE);
  VAR
     LENGTH : INTEGER;
  BEGIN
     LENGTH := 0;
     REPEAT
        LENGTH := LENGTH + 1;
        READ (NAME [LENGTH])
     UNTIL  NAME [LENGTH] = '.';
     WHILE  LENGTH <= NAME_LENGTH  DO
      BEGIN
        NAME [LENGTH] := ' ';
        LENGTH := LENGTH + 1
      END
  END;      { read_name }
```

```
                                    {     הפונקציה READ_DETAILS קוראת את נתוני הסטודנט
                                    .לתוך רשומה חדשה בזיכרון, ומחזירה את המצביע לרשומה זו }

FUNCTION READ_DETAILS : STD_PTR;
   VAR
      TEMP_PTR : STD_PTR;
      LENGTH : INTEGER;
   BEGIN
      OUR_NEW (MEM, TEMP_PTR);
      WITH MEM.SPACE [TEMP_PTR] DO
       BEGIN
         READ_NAME (NAME);
         AVERAGE := 0;
         FOR LENGTH := 1 TO 10 DO
          BEGIN
            READ (EXERCISE [LENGTH]);
            IF EXERCISE [LENGTH] = 'X' THEN
               AVERAGE := AVERAGE + 1
          END;
         READLN (MID_TEST, FINAL_TEST);
         AVERAGE :=
               ROUND (2 * AVERAGE + 0.3 * MID_TEST + 0.5 * FINAL_TEST)
       END;   { with }
      READ_DETAILS := TEMP_PTR
   END;   { read_details }

                                    {     הפרוצדורה PRINT_DETAILS מדפיסה את
                                    .(STUDENT) הנתונים של רשומת הסטודנט }

PROCEDURE PRINT_DETAILS (VAR STUDENT : STUDENT_DETAILS);
   BEGIN
      WITH STUDENT DO
         WRITELN
           (NAME, EXERCISE:10, MID_TEST:8, FINAL_TEST:12, AVERAGE :11)
   END;     { print_details }

                                    {     הפרוצדורה PRINT מדפיסה את נתוני
                                    .הסטודנטים ברשימה ש-LIST מצביע עליה }

PROCEDURE PRINT (LIST : STD_PTR);
   BEGIN
      WRITELN;
      WRITELN ('STUDENT NAME':15, 'EXERCISE':24,
        'MID TEST':12, 'FINAL TEST':12, 'AVERAGE':10);
      WHILE NOT END_OF_LIST (LIST) DO
       BEGIN
         PRINT_DETAILS (MEM.SPACE [LIST]);
         LIST := NEXT (MEM, LIST)
       END;
      WRITELN
   END;     { print }
```

```
                              {      הפרוצדורה PRINT_BEST סורקת רשימה מקושרת,
                                     ש-LIST מצביע עליה, ומדפיסה את שמות הסטודנטים,
                                     שממוצע ציוניהם גבוה או שווה ל-AVERAGE_LEVEL. }

PROCEDURE PRINT_BEST (LIST  : STD_PTR;
                      LEVEL : MARK);
   BEGIN
      WRITELN;
      WRITELN ('STUDENT NAME           AVERAGE':40);
      WHILE  NOT END_OF_LIST (LIST)  DO
       BEGIN
         WITH  MEM.SPACE [LIST]  DO
            IF  AVERAGE >= AVERAGE_LEVEL  THEN
               WRITELN (NAME, AVERAGE:8);
         LIST := NEXT (MEM, LIST)
       END;
      WRITELN
   END;    { print }

                              {      הפרוצדורה FOUND מחזירה מצביע לרשומה, אם
                                     שדה השם בה שווה ל-NAME. היא מחזירה
                                     NULL אם אין ברשימה המקושרת ש-LIST
                                     מצביע עליה רשומה, אשר מקיימת את התנאי לעיל. }

FUNCTION FOUND (VAR NAME : NAME_TYPE;
                    LIST : STD_PTR)   : STD_PTR;
   BEGIN
      IF  NOT END_OF_LIST (LIST)  THEN
         IF  MEM.SPACE [LIST].NAME < NAME  THEN
            FOUND := FOUND (NAME, NEXT (MEM, LIST))
         ELSE
            IF  MEM.SPACE [LIST].NAME = NAME  THEN
               FOUND := LIST
            ELSE                                           { LIST.NAME > NAME }
               FOUND := NULL
      ELSE
         FOUND := NULL
   END;    { found }
```

הפרוצדורה DELETE מוחקת מרשימה ממויינת, {
ש-LIST מצביע עליה, את הרשומה,
} שהשדה NAME שלה שווה ל-STD_NAME.

```
PROCEDURE DELETE (VAR STD_NAME : NAME_TYPE;
                  VAR LIST     : STD_PTR);
   BEGIN
      IF  NOT END_OF_LIST (LIST)  THEN
         IF  MEM.SPACE [LIST].NAME < STD_NAME  THEN
            DELETE (STD_NAME, MEM.SPACE [LIST].NEXT)
         ELSE
            IF  MEM.SPACE [LIST].NAME = STD_NAME  THEN
             BEGIN
               PTR := LIST;
               LIST := NEXT (MEM, LIST);
               OUR_DISPOSE (MEM, PTR)
             END
            ELSE
               WRITELN ('ERROR! - STUDENT NAME NOT FOUND.')
      ELSE
         WRITELN ('ERROR! - STUDENT NAME NOT FOUND.')
   END;     { delete }
```

הפרוצדורה INSERT מכניסה רשומה חדשה {
למקום המתאים ברשימה ממויינת.
NEW_STD מצביע על הרשומה החדשה.
} LIST מצביע על ראש הרשימה.

```
PROCEDURE INSERT (NEW_STD  : STD_PTR;
                  VAR LIST : STD_PTR);
   BEGIN
      IF  NOT END_OF_LIST (LIST)  THEN
         IF  MEM.SPACE [LIST].NAME < MEM.SPACE [NEW_STD].NAME  THEN
            INSERT (NEW_STD, MEM.SPACE [LIST].NEXT)
         ELSE
            IF  MEM.SPACE [LIST].NAME = MEM.SPACE [NEW_STD].NAME  THEN
               WRITELN ('ERROR! - STUDENT ALREADY IN LIST')
            ELSE
             BEGIN
               MEM.SPACE [NEW_STD].NEXT := LIST;
               LIST := NEW_STD;
             END
      ELSE
       BEGIN
         LIST := NEW_STD;
         MEM.SPACE [LIST].NEXT := NULL
       END
   END;     { insert }
```

```
BEGIN     { main - std_list }
   INITIAL_MEM (MEM);                                          { "איפוס הזיכרון" }
   ASSIGN (CLASS, 'CLASS.DAT');
   LIST := LOAD_LIST_FROM_FILE (CLASS); {טעינת הרשימה מהקובץ לזיכרון}
   REPEAT
      READ (COMMAND);                 { קריאת ההוראה לאיחזור מידע מהרשימה }
      CASE  COMMAND  OF                              { הדפסה של כל הרשימה }
         'P' : PRINT (LIST);
         'T' : BEGIN                             { הדפסה של רשומת סטודנט }
                  READ_NAME (STD_NAME);
                  WRITELN;
                  PTR := FOUND (STD_NAME, LIST);
                  IF  NOT  END_OF_LIST (PTR)  THEN
                     PRINT_DETAILS (MEM.SPACE [PTR])
                  ELSE
                     WRITELN ('ERROR! - STUDENT NAME NOT FOUND.')
               END;
         'D' : BEGIN                              { ביטול של רשומת סטודנט }
                  READ_NAME (STD_NAME);
                  WRITELN;
                  DELETE (STD_NAME, LIST)
               END;
         'H' : INSERT (READ_DETAILS, LIST);  { הוספה של רשומת סטודנט }
         'Z' : BEGIN             {  הדפסת השם והציון של הסטודנטים שממוצע }
                                 { ציוניהם גדול או שווה ל-AVERAGE_LEVEL. }
                  READLN (AVERAGE_LEVEL);
                  PRINT_BEST (LIST, AVERAGE_LEVEL)
               END
      END     { case }
   UNTIL  COMMAND = '.';     { הלולאה מתבצעת עד שנקראת הוראת "נקודה". }
   SAVE_LIST_IN_FILE (LIST, CLASS)          { שמירת הרשימה בקובץ CLASS. }
END.
```

<u>הסבר:</u>

הסבר מפורט על הפרוצדורות והפונקציות הקשורות בניהול הזיכרון במערך, תוכל למצוא בתחילת הפרק. שאר הפרוצדורות והפונקציות מוסברות בליווי תיאור סכימטי בתוכנית 1 שבפרק 17.

פרק 17

מבני נתונים דינמיים

מיון של רשימה מקושרת

בפתרון התרגילים שבפרק זה נשתמש בתוכנית המיון שלהלן. תחילה ניתנה התוכנית והפרוצדורות הקשורות בה, ולאחר מכן - הסבר מפורט ותיאור סכימטי של שיטת המיון.
הפרוצדורות למיון הן חלק מתוכנית הישום. הגדרת הרשומה כפי שמובאת להלן הינה דוגמה לרשומה ברשימה מקושרת, בה חייבים להיות שני שדות, אשר בדוגמה זו הם: ELEMENT - שדה מפתח,
NEXT - המצביע לרשומה הבאה.

```
TYPE
   ELEMENT_TYPE = INTEGER;
   NODE_PTR = ^NODE;
   NODE = RECORD
             ELEMENT : ELEMENT_TYPE;
             NEXT    : NODE_PTR
          END;
VAR
   ROOT : NODE_PTR;
   :
   :
PROCEDURE LIST_SORT (VAR FIRST : NODE_PTR);
   VAR
      LAST_UNSORT, { מצביע על הרשומה האחרונה ברשימה שעדיין לא מויינה }
      LAST_REPLACE : NODE_PTR;            { מצביע על הרשומה האחרונה }
                                          { ברשימה שהוחלפה.          }
```

```
PROCEDURE SORT (VAR FIRST : NODE_PTR);
BEGIN
   IF  FIRST^.NEXT <> LAST_UNSORT  THEN
    BEGIN                                   { אם הרשומה הבאה עדיין לא מויינה }
      IF  FIRST^.ELEMENT > FIRST^.NEXT^.ELEMENT  THEN      { השוואת }
                                                           { שדות מפתח. }
       BEGIN                                { החלפת שתי רשומות - ראה ציור }
         LAST_REPLACE := FIRST^.NEXT;
         FIRST^.NEXT := LAST_REPLACE^.NEXT;
         LAST_REPLACE^.NEXT := FIRST;
         FIRST := LAST_REPLACE;
         LAST_REPLACE := FIRST^.NEXT                  { מצביע לרשומה }
                                                      { אחרונה שהוחלפה. }
       END;
      SORT (FIRST^.NEXT)    { קריאה רקורסיבית למיון של המשך הרשימה }
    END    { if }
   ELSE                               { הגענו לסוף הרשימה שיש למיין בשלב זה }
     LAST_UNSORT := LAST_REPLACE             { במעבר הבא יש למיין עד }
                                             { הרשומה האחרונה שהוחלפה. }
END;    { sort }

BEGIN  { list_sort }
   LAST_UNSORT := NIL;                       { ערך התחלתי - סוף הרשימה }
   WHILE  FIRST <> LAST_UNSORT  DO           { הלולאה מתבצעת כל זמן }
    BEGIN                                    { שהמצביע על ראש הרשימה שונה. }
                            { מהמצביע על הרשומה האחרונה שעד אליה יש למיין. }
      LAST_REPLACE := FIRST;                 { ערך התחלתי למצביע }
                                             { הרשומה האחרונה שהוחלפה. }
      SORT (FIRST)                 { קריאה למיון עם מצביע לראש הרשימה }
    END
END;   { list_sort }
 :
 :
BEGIN  { main }
 :
 LIST_SORT (ROOT);       { הפעלת פרוצדורת המיון כחלק מהתוכנית הראשית }
 :
END.
```

<u>הסבר:</u>

התוכנית מורכבת מהפרוצדורות LIST_SORT והפרוצדורה SORT, אשר מקוננות בה. הפרוצדורה LIST_SORT מקבלת מצביע לרשימה מקושרת של רשומות ומחזירה מצביע לרשימה ממויינת לפי שדה מפתח - ELEMENT.

הפרוצדורה פועלת לפי עקרון של מיון בועות. כל מעבר לאורך הרשימה מתבצע באמצעות הפרוצדורה SORT עד הרשומה האחרונה שהוחלפה במעבר הקודם.

א. הפרוצדורה LIST_SORT:

המצביע לרשומה האחרונה שיש למיין מקבל ערך התחלתי NIL, המציין את סוף הרשימה.

הלולאה של המיון מתבצעת כל זמן שהמצביע לראש הרשימה שונה מהמצביע

לרשומה האחרונה שיש למיין:
- הערך ההתחלתי לרשומה האחרונה שהוחלפה (LAST_REPLACE) הוא המצביע לרשומה הראשונה.
- קריאה לפרוצדורה SORT עם מצביע לראש הרשימה.

הלולאה מסתיימת כאשר לא מתבצעת אף לא החלפה אחת (כלומר - הרשימה ממויינת).

ב. הפרוצדורה SORT:

מעבר בודד על הרשימה מתבצע ע"י הפרוצדורה SORT בצורה רקורסיבית, באופן הבא:

כל עוד לא מגיעים לרשומה האחרונה (בפעם הראשונה תהיה זו הרשומה שבסוף הרשימה ולאחר מכן - הרשומה האחרונה שהוחלפה במעבר הקודם).
- משווים את שתי הרשומות הבאות לפי המפתח.
- אם סדר המיון הפוך: - הופכים את סדר הרשומות ברשימה.
  - שומרים את מיקום הרשומה האחרונה שהוחלפה.
  - קוראים ברקורסיה ל-SORT עם מצביע לרשומה הבאה.

אחרת: עדכן את המצביע לרשומה האחרונה (LAST_UNSORT) לפי המצביע לרשומה האחרונה LAST_REPLACE שהוחלפה במעבר זה.

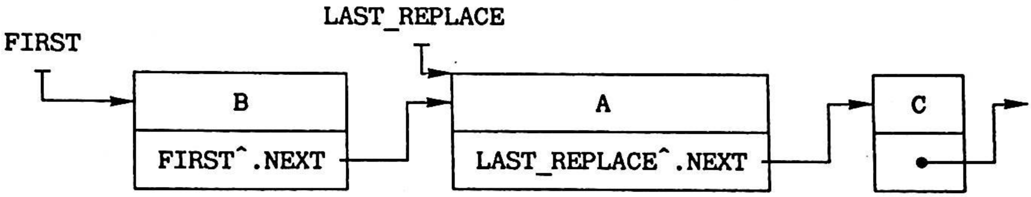
תיאור סכימטי של תהליך המיון

נציג את תהליך ההחלפה של שתי רשומות בשרשרת, אשר שדות המפתח שלהן מסומנות ב-A ו-B.

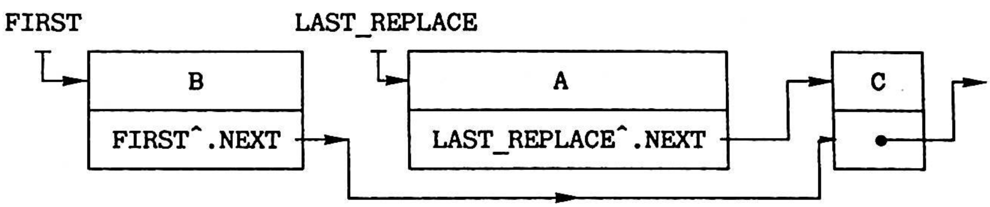
נעקוב אחר תמונת המצב לאחר ביצוע ההוראות השונות של הפרוצדורה SORT.

הערה: המשתנה FIRST אשר בפקודות אלו יכול להיות המצביע אל ראש הרשימה, או להיות השדה NEXT של הרשומה הקודמת. בכל מקרה, אם מחליפים את הרשומה ש-FIRST מצביע עליה ברשומה הבאה, מתעדכן המשתנה ש-FIRST מייצג (מצביע אל ראש הרשימה, או השדה NEXT של הרשומה הקודמת).

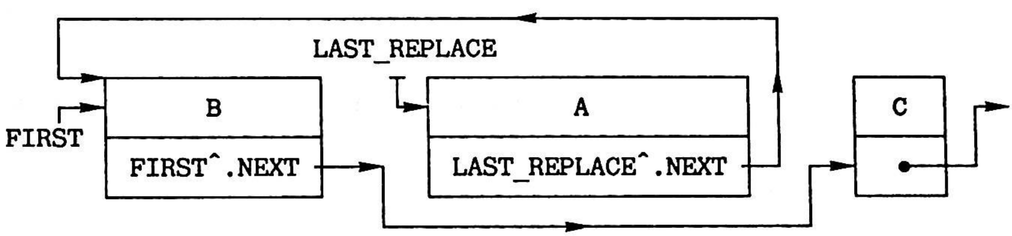
א. מצב התחלתי: LAST_REPLACE := FIRST^.NEXT



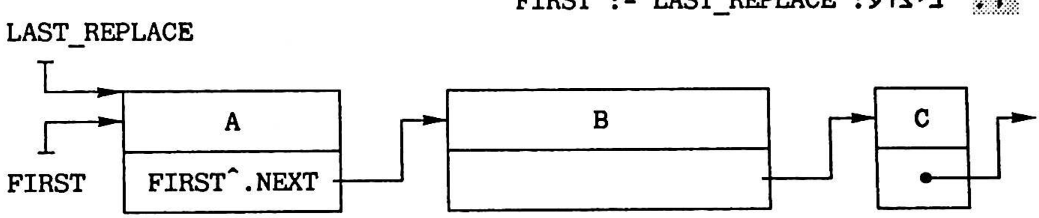
ב. ביצוע: FIRST^.NEXT := LAST_REPLACE^.NEXT



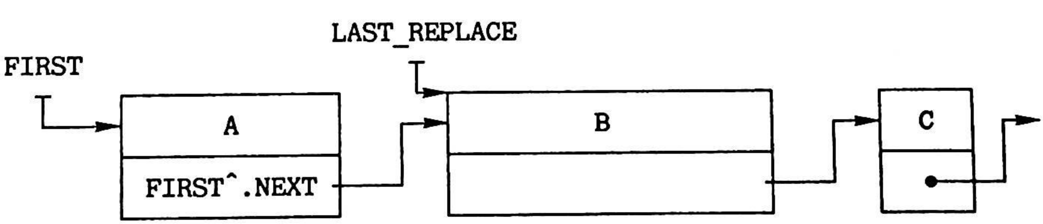
ג. ביצוע: LAST_REPLACE^.NEXT := FIRST



ד. ביצוע: FIRST := LAST_REPLACE

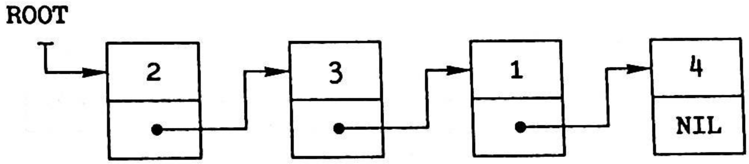


ה. ביצוע: LAST_REPLACE := FIRST^.NEXT



דוגמה למיון:

נתונה רשימה בעלת 4 איברים, אשר ערכיהם מצויינים ברשומות המשורטטות.

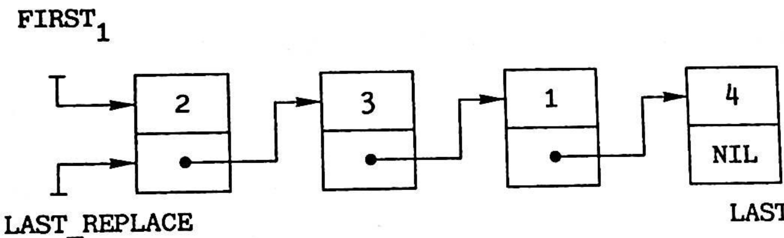


המיון (main) מתחיל בביצוע הפקודה:

LIST_SORT (ROOT);

להלן נתאר באופן סכימטי את שלבי המיון השונים, ובהם מצב המצביעים ובדיקת התנאי. הפרוצדורה SORT פועלת במחזור רקורסיבי.

מצב התחלתי:



LAST_UNSORT := NIL;

התנאי FIRST <> LAST_UNSORT - מתקיים, כי LAST_UNSORT = NIL.

קריאה ל- SORT ($FIRST_1$):

בפעם הראשונה שנכנסים לפרוצדורה SORT נבדק התנאי

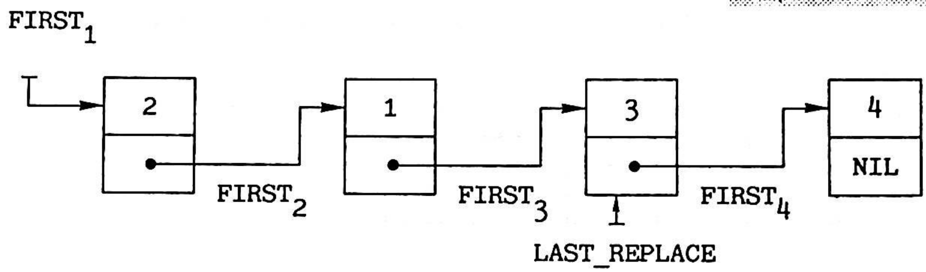
$FIRST_1$^.NEXT <> LAST_UNSORT .

התנאי מתקיים כי LAST_UNSORT = NIL. לכן מתבצעת השוואה בין הערכים 2 ו-3 (שברשומות 1 ו-2) ומתקיים 2 < 3, ולכן אין צורך בהחלפה.
עתה מתבצעת קריאה רקורסיבית ל-SORT ($FIRST_1$^.NEXT):

נשתמש בהמשך בסימון הבא: $FIRST_{n+1}$:= $FIRST_n$^.NEXT

התנאי $FIRST_2$^.NEXT <> LAST_UNSORT מתקיים, ולכן מתבצעת השוואה בין הערכים 1 ל-3. הערך 3 אינו קטן יותר מ-1 ולכן יש צורך בהחלפת איברים.

לאחר ההחלפה נקבל:



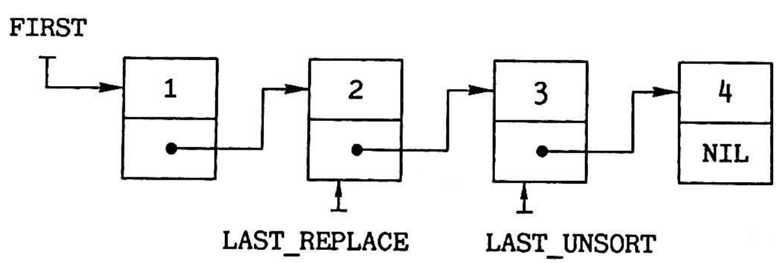
שוב מתבצעת קריאה רקורסיבית ל-SORT (FIRST$_2$^.NEXT). משווים ואין צורך בהחלפה, כי 4 גדול מ-3.

קריאה רקורסיבית נוספת ל-SORT (FIRST$_3$^.NEXT): הפעם מתקיים FIRST$_4$^.NEXT = LAST_UNSORT, ומתבצע עדכון של LAST_UNSORT

LAST_UNSORT := LAST_REPLACE

חוזרים מהקריאות הרקורסיביות להתחלה. עתה FIRST שונה מ-LAST_UNSORT, שעודכן בלולאה ב-LIST_SORT. חוזרים למיין מחדש, הפעם - עם גבולות שונים.

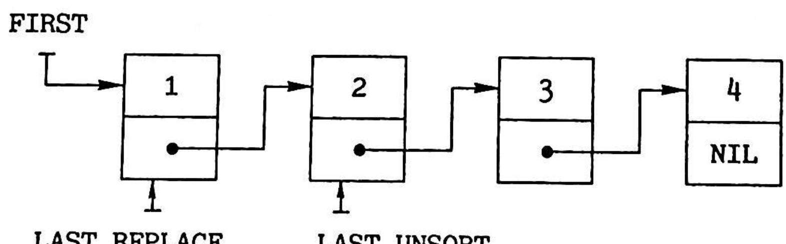
בקריאה הראשונה ל-SORT תתבצע החלפה של שתי הרשומות הראשונות (כי הערך 2 גדול מ-1), ונקבל:



בקריאה השניה ל-SORT לא תתבצע החלפה (כי 2 < 3), ובקריאה השלישית יתבצע עדכון:

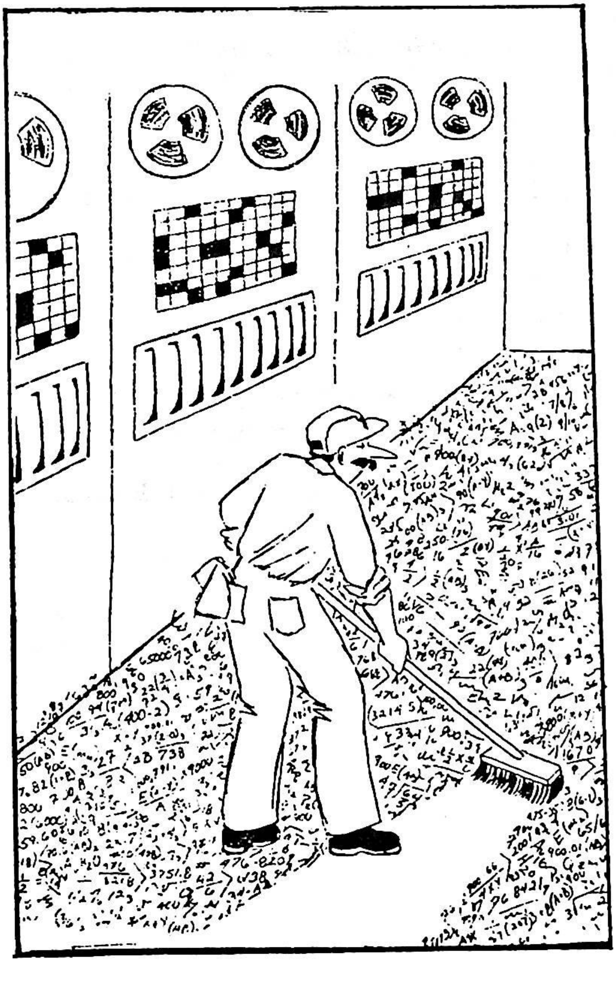
LAST_UNSORT := LAST_REPLACE

בביצוע הלולאה הבאה נקבל:



בקריאה ל-SORT לא תתבצע החלפה, כי 2 < 1, בקריאה השניה יתעדכן LAST_UNSORT ונקבל: LAST_REPLACE =: LAST_UNSORT, וחוזרים מהרקורסיה. בשני המשתנים יהיה עתה ערך של FIRST.

בלולאה הבאה יהיה FIRST=LAST_UNSORT, ולכן תתבצע יציאה מלולאת ה-WHILE ומהפרוצדורה SORT_LIST.



תוכניות

תוכנית 1 (ראה תרגיל תכנות 1 בפרק 16)

```
PROGRAM STD_LIST (INPUT, CLASS, OUTPUT);
CONST
   NAME_LENGTH = 30;                          { אורך מכסימלי של שם }
TYPE
   NAME_TYPE = PACKED ARRAY [1..NAME_LENGTH] OF CHAR;
   MARK = 0..100;
   STD_PTR = ^STUDENT_DETAILS;
   STUDENT_DETAILS = RECORD                   { רשומת פרטי סטודנט }
                     NAME      : NAME_TYPE;   { שם }
                     EXERCISE  : ARRAY [1..10] OF CHAR; { עבודות }
                     MID_TEST,                { ציון במבחן ביניים }
                     FINAL_TEST,              { ציון במבחן סופי }
                     AVERAGE   : MARK;        { ציון ממוצע }
                     NEXT      : STD_PTR      { מצביע לרשומה הבאה }
                   END;
   STD_FILE = FILE OF STUDENT_DETAILS;
VAR
   CLASS     : STD_FILE;                      { קובץ רשומות של פרטי סטודנט }
   LIST,
   PTR       : STD_PTR;                       { מצביע לראש רשימת הסטודנטים }
   COMMAND   : CHAR;
   STD_NAME  : NAME_TYPE;
   AVERAGE_LEVEL : MARK;

{ הפונקציה END_OF_LIST מחזירה ערך TRUE כאשר
  המצביע LIST_PTR הוא NIL, כלומר סוף הרשימה. }

FUNCTION END_OF_LIST (LIST_PTR : STD_PTR) : BOOLEAN;
  BEGIN
     END_OF_LIST := LIST_PTR = NIL
  END; { end_of_list }
```

```
{                    הפונקציה LOAD_LIST_FROM_FILE יוצרת רשימה מקושרת
{ מהרשומות הנמצאות בקובץ CLASS, ומחזירה מצביע לראש הרשימה.

FUNCTION LOAD_LIST_FROM_FILE (VAR CLASS : STD_FILE) : STD_PTR;

                    { CLASS הפונקציה READ_FILE קוראת רשומה אחת מקובץ
                    { לתוך רשימה מקושרת, ומחזירה מצביע לאותה רשומה.

   FUNCTION READ_FILE : STD_PTR;
      VAR
         TEMP_PTR : STD_PTR;
      BEGIN
         IF  EOF (CLASS)  THEN                          { אם סוף הקובץ - }
            READ_FILE := NIL                            {       סוף הרשימה. }
         ELSE                                        { אחרת - לא סוף הקובץ }
          BEGIN
            NEW (TEMP_PTR);             { הקצאת מקום בזיכרון לרשומה חדשה. }
            READ (CLASS, TEMP_PTR^);       { קריאת רשומה מקובץ CLASS אל }
                            { רשומה בזיכרון, שעליה מצביע ^TEMP_PTR. }
            TEMP_PTR^.NEXT := READ_FILE;             { שרשור השדה NEXT }
                         { ע"י קריאה רקורסיבית לפונקציה READ_FILE. }
            READ_FILE := TEMP_PTR      { החזרת ערך המצביע לרשומה החדשה }
          END
      END; { read_file }

   BEGIN { load_list_from_file }
      RESET (CLASS);
      LOAD_LIST_FROM_FILE := READ_FILE            { החזרת המצביע לרשומה }
   END;  { load_list_from_file }                  {    הראשונה ברשימה. }

        {  ,הפרוצדורה SAVE_LIST_IN_FILE כותבת לקובץ CLASS את הרשומות
                 { שנמצאות ברשימה המקושרת ש-LIST_PTR מצביע עליה.

PROCEDURE SAVE_LIST_IN_FILE (LIST_PTR : STD_PTR;
                             VAR CLASS : STD_FILE);

   BEGIN
      REWRITE (CLASS);
      WHILE  NOT END_OF_LIST (LIST_PTR)  DO      {     לולאה - כל זמן שלא }
                                                 { מגיעים לסוף הרשימה. }
       BEGIN
         WRITE (CLASS, LIST_PTR^);           { כתיבת רשומה ש-LIST_PTR מצביע }
                                             {       עליה לקובץ CLASS. }
         LIST_PTR := LIST_PTR^.NEXT     {הזזת המצביע לרשומה הבאה ברשימה }
       END;
      CLOSE (CLASS)
   END; { save_list_in_file }
```

```
{ הפרוצדורה READ_NAME קוראת מחרוזת של תווים
  לתוך מערך NAME. סוף המחרוזת מסומן ע"י נקודה. }
PROCEDURE READ_NAME (VAR NAME : NAME_TYPE);
   VAR
      LENGTH : INTEGER;
   BEGIN
     LENGTH := 0;
     REPEAT
        LENGTH := LENGTH + 1;
        READ (NAME [LENGTH])
     UNTIL  NAME [LENGTH] = '.';             { הנקודה מסמנת את סוף המחרוזת }
     WHILE  LENGTH <= NAME_LENGTH  DO
      BEGIN                                   { מילוי רווחים עד סוף המערך }
        NAME [LENGTH] := ' ';
        LENGTH := LENGTH + 1
      END
 END; { read_name }

{ הפונקציה READ_DETAILS קוראת את נתוני הסטודנט
  לתוך רשומה חדשה בזיכרון ומחזירה את המצביע לרשומה זו. }
FUNCTION  READ_DETAILS : STD_PTR;
   VAR
      TEMP_PTR : STD_PTR;
      LENGTH : INTEGER;
  BEGIN
     NEW (TEMP_PTR);                    { TEMP_PTR מצביע לרשומה חדשה בזיכרון }
     WITH  TEMP_PTR^ DO   { מכאן ההתייחסות היא לשדות של הרשומה החדשה }
      BEGIN
        READ_NAME (NAME);                      { קריאת שם הסטודנט לשדה NAME }
        AVERAGE := 0;
        FOR  LENGTH := 1  TO  10  DO
         BEGIN                                 { קריאת הציונים של 10 תרגילים }
           READ (EXERCISE [LENGTH]);
           IF  EXERCISE [LENGTH] = 'X'  THEN
              AVERAGE := AVERAGE + 1
         END;
       READLN (MID_TEST, FINAL_TEST);                  { קריאת ציוני הבחינות }
       AVERAGE :=                                            { חישוב הממוצע. }
            ROUND (2 * AVERAGE + 0.3 * MID_TEST + 0.5 * FINAL_TEST)
     END;
     READ_DETAILS := TEMP_PTR                          { החזרת המצביע לרשומה }
 END; { read_details }
```

```
                          {     הפרוצדורה PRINT_DETAILS מדפיסה את
                               הנתונים של רשומת הסטודנט (STUDENT). }

PROCEDURE PRINT_DETAILS (VAR STUDENT : STUDENT_DETAILS);
   BEGIN
      WITH  STUDENT  DO
         WRITELN
           (NAME, EXERCISE:10, MID_TEST:8, FINAL_TEST:12, AVERAGE :11)
   END; { print_details }

                          {       הפרוצדורה PRINT מדפיסה את נתוני
                               הסטודנטים ברשימה ש-LIST מצביע עליה. }

PROCEDURE PRINT (LIST : STD_PTR);
   BEGIN
      WRITELN;
      WRITELN ('STUDENT NAME':15, 'EXERCISE':24,
        'MID TEST':12, 'FINAL TEST':12, 'AVERAGE':10);
      WHILE  NOT END_OF_LIST (LIST)  DO
       BEGIN
         PRINT_DETAILS (LIST^);      { הדפסת הרשומה ש-LIST מצביע עליה }
         LIST := LIST^.NEXT                { הזזת המצביע לרשומה הבאה }
       END;
      WRITELN
   END; { print }

                  {        הפרוצדורה PRINT_BEST סורקת רשימה מקושרת,
                     ש-LIST מצביע עליה, ומדפיסה את שמות הסטודנטים
                     שממוצע ציוניהם גבוה או שווה ל-AVERAGE_LEVEL. }

PROCEDURE PRINT_BEST (LIST : STD_PTR;
                      AVERAGE_LEVEL : MARK);

   BEGIN
      WRITELN;
      WRITELN ('STUDENT NAME            AVERAGE':40);
      WHILE  NOT END_OF_LIST (LIST)  DO
       BEGIN
         WITH  LIST^  DO
            IF  AVERAGE >= AVERAGE_LEVEL  THEN
               WRITELN (NAME, AVERAGE:8);
         LIST := LIST^.NEXT
       END;
      WRITELN
   END; { print_best }
```

```
הפרוצדורה FOUND מחזירה מצביע לרשומה אם }
שדה השם בה שווה ל-NAME. היא מחזירה
NIL אם אין רשומה ברשימה המקושרת ש-LIST
מצביע עליה, אשר מקיימת את התנאי לעיל. {

FUNCTION FOUND (VAR NAME : NAME_TYPE;
                    LIST : STD_PTR)   : STD_PTR;
   BEGIN
      IF  NOT END_OF_LIST (LIST)  THEN                  { אם לא סוף הרשימה }
         IF  LIST^.NAME < NAME  THEN   { השדה NAME ברשומה קטן מ-NAME }
            FOUND := FOUND (NAME, LIST^.NEXT)          { קריאה רקורסיבית }
                                                { לחיפוש בהמשך הרשימה. }
         ELSE
            IF  LIST^.NAME = NAME  THEN
               FOUND := LIST                                  { הרשומה נמצאה }
            ELSE        { LIST^.NAME > NAME }
               FOUND := NIL          { NAME לא נמצא כי הרשימה ממויינת }
      ELSE
         FOUND := NIL
   END; { found }

פרוצדורה DELETE מוחקת מרשימה ממויינת, }
ש-LIST מצביע עליה, את הרשומה
שהשדה NAME שלה שווה ל-STD_NAME. {

PROCEDURE DELETE (VAR STD_NAME : NAME_TYPE;
                  VAR LIST     : STD_PTR);
  BEGIN
     IF  NOT END_OF_LIST (LIST)  THEN
        IF  LIST^.NAME < STD_NAME  THEN
           DELETE (STD_NAME, LIST^.NEXT)
        ELSE
           IF  LIST^.NAME = STD_NAME  THEN                    { הרשומה נמצאה }
            BEGIN
              PTR := LIST;
              LIST := LIST^.NEXT;
              DISPOSE (PTR)
            END
           ELSE
              WRITELN ('ERROR! - STUDENT NAME NOT FOUND.')
    ELSE
       WRITELN ('ERROR! - STUDENT NAME NOT FOUND.')
 END; { delete }
```

```
{ הפרוצדורה INSERT מכניסה רשומה חדשה
  למקום המתאים ברשימה ממויינת.
  NEW_STD מצביע על הרשומה החדשה.
  LIST מצביע על ראש הרשימה. }

PROCEDURE INSERT (NEW_STD : STD_PTR;
                  VAR LIST : STD_PTR);
   BEGIN
      IF  NOT END_OF_LIST (LIST)  THEN
         IF  LIST^.NAME < NEW_STD^.NAME  THEN
            INSERT (NEW_STD, LIST^.NEXT)
         ELSE
            IF  LIST^.NAME = NEW_STD^.NAME  THEN
               WRITELN ('ERROR! - STUDENT ALLREADY IN LIST')
                                                  { ברשימה נמצא שם זהה }
            ELSE
             BEGIN
               NEW_STD^.NEXT := LIST;
               LIST := NEW_STD
             END
      ELSE                                                 { סוף הרשימה }
       BEGIN                                         { הוספה בסוף הרשימה }
         LIST := NEW_STD;
         LIST^.NEXT := NIL
       END
   END; { insert }


BEGIN { main }
   ASSIGN (CLASS, 'CLASS.DAT');
   LIST := LOAD_LIST_FROM_FILE (CLASS); {טעינת הרשימה מהקובץ לזיכרון}
   REPEAT
      READ (COMMAND);                                      { קריאת ההוראה }
      CASE  COMMAND  OF
         'P' : PRINT (LIST);                           { הדפסת כל הרשימה }
         'T' : BEGIN                                 { הדפסת רשומת סטודנט }
                  READ_NAME (STD_NAME);                { קריאת שם סטודנט }
                  WRITELN;
                  PTR := FOUND (STD_NAME, LIST); {חיפוש של השם ברשימה}
                  IF  NOT  END_OF_LIST (PTR)  THEN { אם נמצאה הרשומה }
                     PRINT_DETAILS (PTR^)        {- הדפסת פרטי הסטודנט.}
                  ELSE              { הסטודנט המבוקש לא נמצא ברשימה }
                     WRITELN ('ERROR! - STUDENT NAME NOT FOUND.')
               END;
         'D' : BEGIN                                 { ביטול רשומת סטודנט }
                  READ_NAME (STD_NAME);                  { קרא שם סטודנט }
                  WRITELN;
                  DELETE (STD_NAME, LIST)      { מחיקת הסטודנט מהרשימה }
               END;
         'H' : INSERT (READ_DETAILS, LIST);          { הוספת רשומת סטודנט
                                                        חדשה לרשימה. }
```

```
        'Z' : BEGIN     { הדפסת שם וציון של הסטודנטים שממוצע ציוניהם }
                        { גדול או שווה ל-AVERAGE_LEVEL. }
                  READLN (AVERAGE_LEVEL);
                  PRINT_BEST (LIST, AVERAGE_LEVEL)
               END
     END    { case }
   UNTIL  COMMAND = '.';      { הלולאה מתבצעת עד שנקראת הוראת "נקודה" }
   SAVE_LIST_IN_FILE (LIST, CLASS)      { שמירת הרשימה בקובץ CLASS. }
END.
```

הדגמה של פרוצדורות

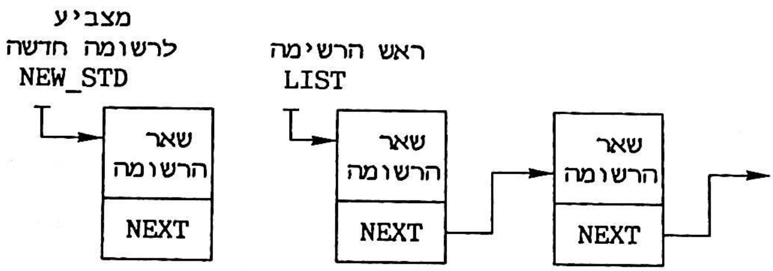
בתוכנית זו ובפרוצדורות שלהלן, סדר המיון הינו לפי שדה NAME, אשר ברשומת הסטודנט STUDENT_DETAILS.

הפרוצדורה INSERT

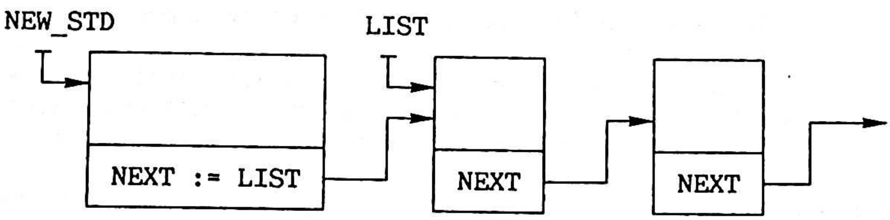
מטרה: הכנסה של רשומה חדשה לרשימה מקושרת, לפי סדר המיון.

הסבר:

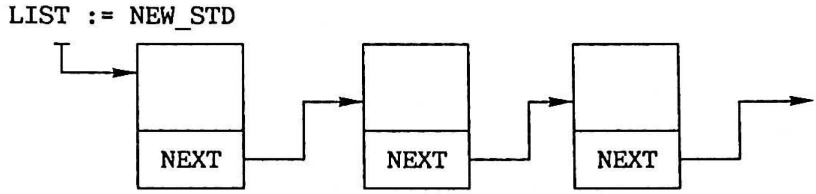
נבחן מקרה פרטי בו הרשומה צריכה להכנס בראש הרשימה.



השדה NEW_STD^.NEXT מקבל את ערך המצביע לראש הרשימה (LIST).



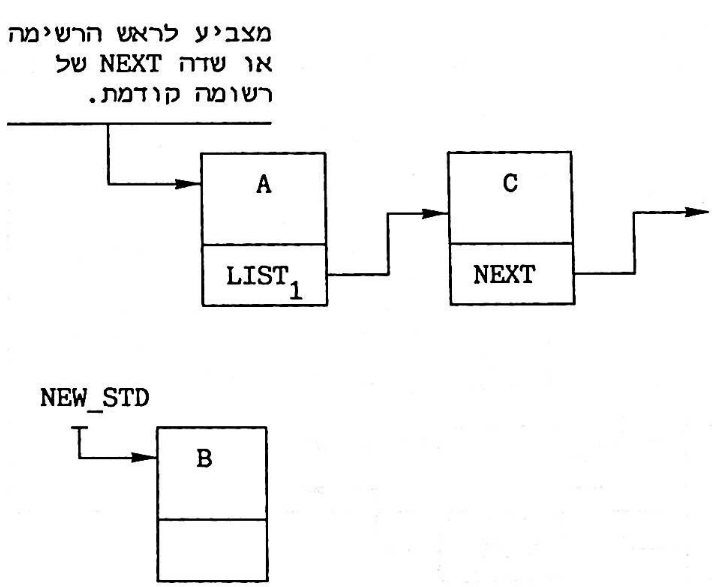
עדכון המצביע לראש הרשימה.



נשתמש במקרה פרטי זה בצורה הבאה:
כאשר שדה NAME של הרשומה, ש-LIST מצביע עליה, קודם לפי סדר המיון לשדה השם ברשומה החדשה, מבצעים קריאה רקורסיבית ל-INSERT עם מצביע לרשומה הבאה.

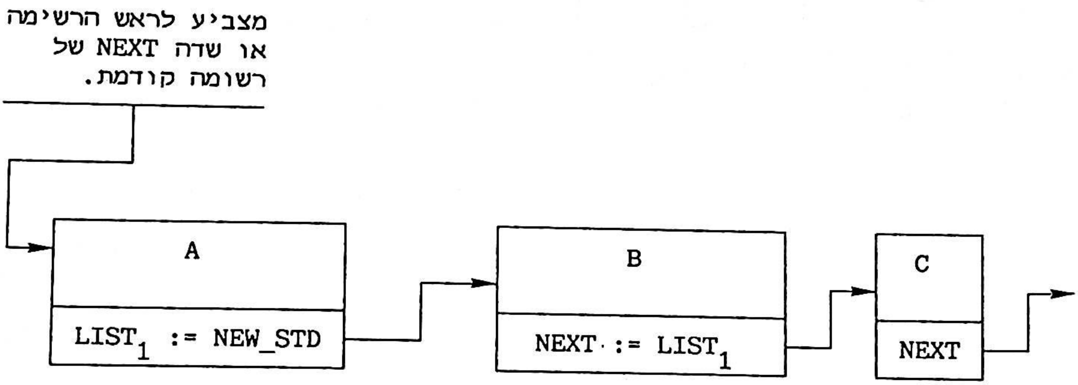
בתהליך הרקורסיה נוכל לזהות שלושה מצבים:
א. סוף הרשימה: כל הרשומות ברשימה, קודמות לרשומה החדשה לפי סדר המיון - יש להוסיף את הרשומה החדשה בסוף הרשימה.
ב. רשומה ששדה השם בה שווה לשדה השם ברשומה החדשה: זוהי טעות, כי הרשומה כבר נמצאת ברשימה ואנו לא מתירים רשומות כפולות בתרגיל זה.
ג. הרשומה הבאה ברשימה צריכה להיות בסדר המיון לאחר הרשומה החדשה: מקרה זה זהה למקרה הפרטי. LIST מצביע כעת לרשומה הבאה אשר גבוהה בסדר המיון מהרשומה החדשה.

תיאור סכימטי של הוספת הרשומה B בין A לבין C:



א. NEW_STD^.NEXT מקבל את ערכו של $LIST_1$ ומצביע לרשומה C.

ב. עדכון ערכו של $LIST_1$ למצביע על רשומה B.

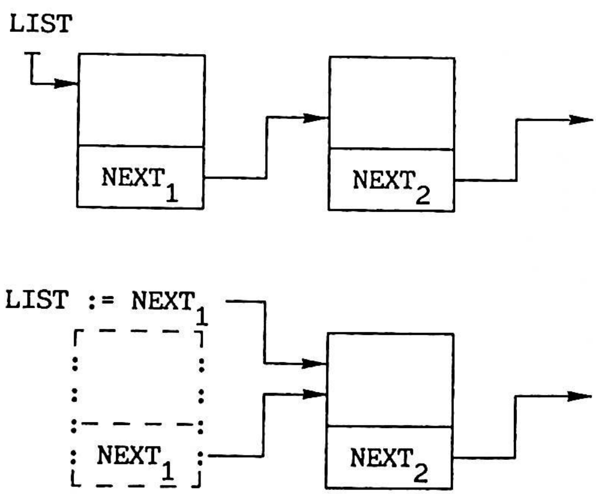


הפרוצדורה DELETE

מטרה: מחיקה של רשומה מרשימה ממויינת, לפי שדה השם.

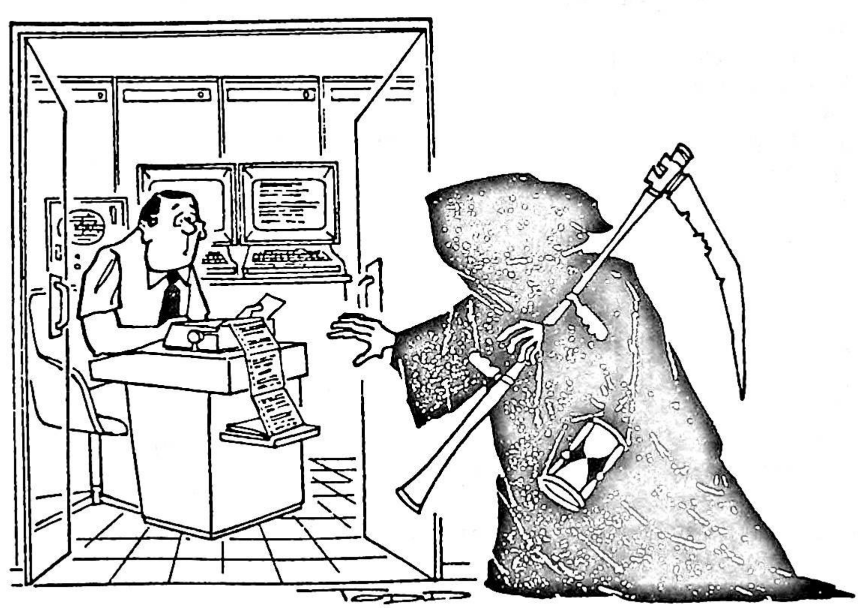
הסבר:

נניח מקרה פרטי בו צריך למחוק את הרשומה מראש הרשימה: הפרוצדורה מעדכנת את המצביע לראש הרשימה וכותבת בו את ערך השדה NEXT של הרשומה שנמחקה.



מקבלים את המצביע לראש הרשימה ומשתמשים בו כדי להתקדם ברשימה ע"י קריאות רקורסיביות לפורוצדורה DELETE. בכל פעם הקריאה תיעשה עם מצביע לרשומה הבאה, עד ששדה השם ברשומה הנוכחית יהיה שווה לשם ברשומה שיש למחוק. בשלב זה מגיעים למצב שתואר בדוגמה לעיל.

שים לב: המשתנה LIST מועבר ע"י VAR, ולכן כאשר מעדכנים אותו בקריאה רקורסיבית מעדכנים גם את השדה NEXT של הרשומה הקודמת.
כאשר מגיעים לרשומה ששדה השם שלה צריך להיות לפי סדר המיון לאחר השם שמחפשים - הרשומה שמחפשים לא נמצאת ברשימה ולכן מדפיסים הודעת שגיאה.



"הירגע, באתי רק לקחת מסוף ישן".

©DATAMATION

תוכנית 2 (ראה תרגיל תכנות 2 בפרק 16)

```
PROGRAM M_A_T_R_I_X (INPUT, OUTPUT);
CONST
   MAX_SIZE = 10;                              { מספר שורות מכסימלי }
TYPE
   NEXT_VALUE = ^VALUE;
   VALUE = RECORD                  { רשומה של ערך השונה מאפס במטריצה }
             COLUMN,                                    { מספר הטור }
             VAL       : INTEGER;                       { ערך }
             NEXT      : NEXT_VALUE    { מצביע לערך הבא באותה שורה }
           END;
   MATRIX = RECORD                               { רשומת מטריצה }
             ROWS,                               { מספר השורות במטריצה }
             COLUMNS   : 1..MAX_SIZE;            { מספר העמודות במטריצה }
             START_ROW : ARRAY [1..MAX_SIZE] OF NEXT_VALUE { מערך
           END;                                  מצביעים לתחילת השורות. }
VAR
   M : MATRIX;

PROCEDURE READ_MATRIX (VAR M : MATRIX);          { קריאת נתונים למטריצה }
   VAR
      TEMP   : NEXT_VALUE;
      VALUE,
      I, J   : INTEGER;
      FIRST  : BOOLEAN;
  BEGIN
     WITH M DO
      BEGIN
        READLN (ROWS, COLUMNS);                  { קריאת מידות המטריצה. }
        FOR I := 1 TO ROWS DO                    { לולאה על שורות המטריצה }
         BEGIN
           FIRST := TRUE;                { מצביע על קריאת ערך ראשון בשורה }
           FOR J := 1 TO COLUMNS DO        { לולאה על עמודות המטריצה }
            BEGIN
              READ (VALUE);                                { קריאת ערך }
              IF VALUE <> 0 THEN
               BEGIN                              { הערך שנקרא שונה מאפס }
                 IF FIRST THEN
                  BEGIN
                    FIRST := FALSE;                     { ערך ראשון בשורה. }
                    NEW (START_ROW [I]);               { מצביע לתחילת השורה. }
                    TEMP := START_ROW [I]
                  END
                 ELSE                     { נקרא ערך והוא אינו הראשון בשורה }
                  BEGIN
                    NEW (TEMP^.NEXT);                   { מצביע לרשומה הבאה }
                    TEMP := TEMP^.NEXT
                  END;
```

```
             TEMP^.VAL := VALUE;                  {.הכנסת הערך לרשומה}
             TEMP^.COLUMN := J                    { .הכנסת מספר הטור}
           END   { if value }
        END;   { for j }
       IF  FIRST  THEN
                          { TRUE אם לאחר הקריאה משתנה FIRST עדיין}
                               {.לא קראנו אף לא ערך אחד השונה מאפס -}
         START_ROW [I] := NIL { בשורה זו אין ערכים השונים מאפס}
       ELSE
                                     { ציון של סוף רשימת הערכים}
         TEMP^.NEXT := NIL;          {     .I השונים מאפס בשורה}
       READLN                                       { שורה חדשה}
     END    { for i }
   END    { with m }
END;    { read_matrix }

       { הדפסת המטריצה M. הפרמטר MODE קובע אם יודפסו כל איברי}
       {.המטריצה (כולל אפסים), או שיודפסו רק הערכים השונים מאפס}

PROCEDURE PRINT_MATRIX (VAR M    : MATRIX;
                            MODE : CHAR);
   VAR
      TEMP : NEXT_VALUE;
      I, J : INTEGER;
   BEGIN
      WITH  M  DO
       BEGIN
         FOR  I := 1  TO  ROWS  DO                 { לולאה על מספר השורות}
          BEGIN
            TEMP := START_ROW [I];                    { מצביע לתחילת השורה}
            J := 1;
            WHILE  TEMP <> NIL  DO                    { לולאה עד סוף רשימת}
             BEGIN                                    {  .הערכים בשורה זו}
               WHILE  (J < TEMP^.COLUMN) AND (MODE = 'B')  DO   { אם}
                BEGIN                                     { MODE = 'B' }
                  WRITE (0:5);                  { מדפיסים איברים עד האיבר}
                  J := J + 1                    {           .הבא השונה מאפס}
                END;                       { יציאה מהלולאה אם הגענו לאיבר}
                                           {   .MODE='A'-השונה מאפס או ש}
               J := J + 1;
               WRITE (TEMP^.VAL:5);           {       .הדפסת איבר השונה מאפס}
               TEMP := TEMP^.NEXT             {.מעבר לאיבר הבא השונה מאפס}
             END;   { while temp }
            WHILE  (J <= COLUMNS) AND (MODE = 'B')  DO
             BEGIN      { אם MODE = 'B' אז הדפס אפסים עד סוף השורה}
               WRITE (0:5);
               J := J + 1
             END;
            WRITELN                                         { שורה חדשה}
          END  { for i }
       END;   { with m }
```

```
      WRITELN
    END;     { print_matrix }

          { הפונקציה מחזירה את סכום ריבועי האיברים של אלכסון המטריצה }

  FUNCTION SQUARS (VAR M : MATRIX) : INTEGER;
    VAR
      TEMP : NEXT_VALUE;
      I,
      SUM  : INTEGER;

    FUNCTION MIN (A,        { הפונקציה מחזירה את המספר הקטן מבין A או B }
                  B  : INTEGER) : INTEGER;
      BEGIN    { min }
        IF  A < B  THEN
          MIN := A
        ELSE
          MIN := B
      END;    { min }

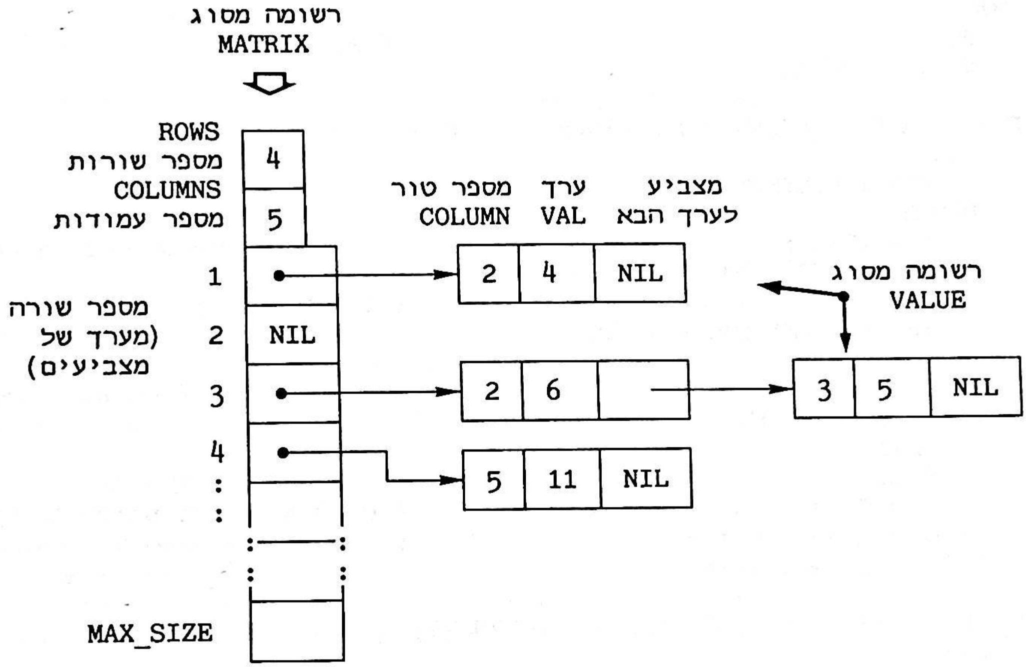
    BEGIN  { squars }
      SUM := 0;
      WITH  M  DO
       BEGIN
         FOR  I := 1  TO  MIN (ROWS, COLUMNS)  DO      { לולאה מ-1 עד }
          BEGIN              { המינימום שבין מספר העמודות למספר השורות. }
            TEMP := START_ROW [I];                { מצביע לאיבר הראשון }
                                                  { ברשימה, בשורה I. }
            WHILE  (TEMP <> NIL) AND (TEMP^.COLUMN < I)  DO
               TEMP := TEMP^.NEXT;
               { התקדמות ברשימת הערכים בשורה I עד למציאת איבר הנמצא
                 על האלכסון (טור = עמודה), או עד לאיבר הבא אחריו. אם
                 הרשימה בשורה זו ריקה לא תתבצע הלולאה כי: TEMP = NIL. }
            IF  TEMP^.COLUMN = I  THEN    { אם נמצא האיבר שעל האלכסון }
               SUM := SUM + SQR (TEMP^.VAL)   { הוספה לסכום הריבועים }
          END
      END;     { with m }
      SQUARS := SUM
   END;     { squars }

BEGIN  { main }
   READ_MATRIX (M);                                  { קריאת המטריצה. }
   PRINT_MATRIX (M, 'A');                  { הדפסת המטריצה ללא אפסים. }
   PRINT_MATRIX (M, 'B');                  { הדפסת המטריצה, כולל אפסים. }
   WRITELN (SQUARS(M), ' סכום הריבועים של איברי האלכסון: ')
END.
```

הסבר:

נתונה המטריצה הבאה:

0	4	0	0	0
0	0	0	0	0
0	6	5	0	0
0	0	0	0	11



תוכנית 3 (ראה תרגיל תכנות 3 בפרק 16)

```
PROGRAM POINTERS_POLYNOM (INPUT, OUTPUT);
TYPE
   POLYNOM = ^PAIR;
   PAIR = RECORD
             FACTOR,                                  { מקדם }
             POWER    : INTEGER;                      { חזקה }
             NEXT     : POLYNOM             { מצביע לאיבר הבא }
          END;
VAR
   P,
   Q  : POLYNOM;

FUNCTION READ_POLYNOM : POLYNOM;
   VAR
      PTR : POLYNOM;
   BEGIN
      NEW (PTR);                              { הקצאת מקום לרשומה חדשה }
      WITH  PTR^  DO
         READLN (FACTOR, POWER);              { קריאת נתונים (מקדם, חזקה) }
      IF  PTR^.FACTOR = 0  THEN                              { סוף קלט }
       BEGIN
         DISPOSE (PTR);                                   { שחרור רשומה }
         PTR := NIL
       END
      ELSE
         PTR^.NEXT := READ_POLYNOM;           { קריאה רקורסיבית לשאר הקלט }
      READ_POLYNOM := PTR                     {    החזרת ערך המצביע לרשומה }
   END;     { read_polynom }

PROCEDURE PRINT_POLYNOM (TEMP : POLYNOM);
   VAR
      FIRST : BOOLEAN;
   BEGIN
      IF  TEMP = NIL  THEN                    { אם הרשימה ריקה הדפס אפס (0) }
         WRITELN (0)
      ELSE
       BEGIN
         FIRST := TRUE;                       {             איבר ראשון. }
         WHILE  TEMP <> NIL  DO               { לולאה עד סוף הרשימה }
          BEGIN
            WITH  TEMP^  DO                   { משלב זה - ההתייחסות היא לרשומה }
             BEGIN                            {          ש-TEMP^ מצביע עליה. }
               IF  FACTOR < 0  THEN
                  WRITE (' - ')                               { המקדם שלילי }
               ELSE                                           { המקדם חיובי }
                  IF  NOT FIRST  THEN         { אם אין זה איבר ראשון }
                     WRITE (' + ');           {            - הדפס '+'. }
               FIRST := FALSE;
```

```
            IF  ABS (FACTOR) <> 1 THEN { 1 או -1 :אם המקדם שונה מ }
                WRITE (ABS (FACTOR));           { - הדפס אותו. }
            IF  POWER <> 0  THEN               { אם החזקה שונה מאפס }
               WRITE ('X');                     { - הדפס 'X'. }
            IF NOT (POWER IN [0, 1]) THEN {1 אם החזקה שונה מ-0 או }
               WRITE ('**', POWER:1)            { - הדפס אותה.}
          END;   { with temp^ }
         TEMP := TEMP^.NEXT                    { עבור לרשומה הבאה }
        END;   { while temp }
       WRITELN
      END   { if }
   END;    { print_polynom }
```

{ פרוצדורה לחיבור מקדמים של 2 איברים בפולינום, אשר PTR1 ו-PTR2
מצביעים עליהם. התוצאה תהיה במקדם של הרשומה ש-PTR1 מצביע עליה. }

```
PROCEDURE ADD_FACTOR (PTR1,
                      PTR2  : POLYNOM);
   BEGIN
      PTR1^.FACTOR := PTR1^.FACTOR + PTR2^.FACTOR
   END;    { add_factor }
```

{ מחיקת רשומה (איבר), ש-PTR מצביע עליה, מהרשימה }

```
PROCEDURE DELETE_PAIR (VAR PTR : POLYNOM);
   VAR
      TEMP : POLYNOM;
   BEGIN
      TEMP := PTR;               {            שמירה זמנית של המצביע. }
      PTR  := PTR^.NEXT;         {    עדכון המצביע לרשומה הבאה ברשימה. }
      DISPOSE (TEMP)             {.שחרור שטח בזיכרון של הרשומה שנמחקה }
   END;     { delete_pair }
```

{ חיבור שתי רשימות לרשימה אחת חדשה }

```
PROCEDURE CONNECT_POLYNOMS (SRC_1,
                            SRC_2  : POLYNOM;
                        VAR DEST   : POLYNOM);
   VAR
      TEMP : POLYNOM;
   BEGIN
      IF  (SRC_1 <> NIL) OR (SRC_2 <> NIL)  THEN
       BEGIN
         NEW (DEST)                   { אחת הרשימות לפחות אינה ריקה }
         TEMP := DEST;
         REPEAT                  {     כל זמן שהמצביע אל הרשימה אינו }
           WHILE  SRC_1 <> NIL DO {.מצביע לסופה - מבצעים את הלולאה }
            BEGIN            { העתקת רשומה מרשימת המקור לרשימת התוצאה }
             TEMP^ := SRC_1^;
```

```
            IF  TEMP^.NEXT <> NIL  THEN                { אם לא סוף הרשימה }
             BEGIN
              NEW (TEMP^.NEXT);          { הקצאת מקום בזיכרון לרשומה חדשה. }
              TEMP  := TEMP^.NEXT        { קידום המצביע ברשימת התוצאה. }
             END;
            SRC_1  := SRC_1^.NEXT              { קידום המצביע ברשימת המקור }
           END;    { while }
          IF  SRC_2 <> NIL  THEN
           BEGIN                                       { רשימה 2 אינה ריקה }
            SRC_1  := SRC_2;        { SRC_1 יצביע על תחילת הרשימה השניה }
            SRC_2  := NIL;
                  { SRC_2 מקבל את הערך NIL כדי שמעבר זה לא יתבצע שנית }
            NEW (TEMP^.NEXT);            { הקצאת מקום בזיכרון לרשומה חדשה. }
            TEMP  := TEMP^.NEXT          { קידום המצביע של הרשימה החדשה. }
           END
             { בשלב זה מצביע SRC_1 על רשימה מספר 2. כלומר - הרשימה
               הראשונה הועתקה כולה לרשימת התוצאה. ואז SRC_2:=NIL.
               בפעם השניה שיתבצע הקטע התחום על ידי REPEAT,
               גם SRC_1:=NIL, והלולאה תפסק לאחר ששתי הרשימות הועתקו,
               בזו אחר זו, לרשימת התוצאה. }
        UNTIL  SRC_1 = NIL
      END    { if scr_1 }
     ELSE                                           { שתי הרשימות ריקות }
        DEST := NIL
  END;     { connect_polynoms }

                                              { מיון רשימה (לפי דוגמא 1) }

PROCEDURE LIST_SORT (VAR FIRST : POLYNOM);
   VAR
      LAST_UNSORT,
      LAST_REPLACE : POLYNOM;

   PROCEDURE SORT (VAR FIRST : POLYNOM);
   BEGIN
     IF  (FIRST^.NEXT <> LAST_UNSORT)  THEN
      BEGIN
        IF  FIRST^.POWER > FIRST^.NEXT^.POWER  THEN
         BEGIN
           LAST_REPLACE        := FIRST^.NEXT;
           FIRST^.NEXT         := LAST_REPLACE^.NEXT;
           LAST_REPLACE^.NEXT  := FIRST;
           FIRST               := LAST_REPLACE;
           LAST_REPLACE        := FIRST^.NEXT
         END;
        SORT (FIRST^.NEXT)
      END    { if (first^...}
     ELSE
        LAST_UNSORT := LAST_REPLACE
  END;     { sort }
```

```
BEGIN  { list_sort }
   LAST_UNSORT := NIL;
   WHILE  FIRST <> LAST_UNSORT  DO
    BEGIN
      LAST_REPLACE := FIRST;
      SORT (FIRST)
    END
END;    { list_sort }
```

{ פרוצדורה לחיבור של חזקות שוות }

```
FUNCTION ADD_SAME_POWERS (P : POLYNOM) : POLYNOM;
   VAR
      DISP_PTR : POLYNOM;
   BEGIN
      IF  (P <> NIL)  THEN
       BEGIN
         WHILE  (P^.NEXT <> NIL) AND (P^.POWER = P^.NEXT^.POWER)  DO
          BEGIN                                          { אם החזקות שוות }
            ADD_FACTOR (P, P^.NEXT);                     { חבר מקדמים. }
            DELETE_PAIR (P^.NEXT)                        { מחק את הרשומה השניה מבין }
                                                         { אילו שחיברת את מקדמיהם. }
          END;
         IF  P^.FACTOR = 0  THEN
          BEGIN                                          { מקדם שווה אפס. }
            DISP_PTR := P;                               { שמירת ערך המצביע. }
            P := ADD_SAME_POWERS (P^.NEXT);              { הערך החדש של המצביע }
                                                         { יתקבל ע"י קריאה רקורסיבית לפונקציה. }
            DISPOSE (DISP_PTR)                           { שחרור רשומה קודמת עם מקדם 0 }
          END
         ELSE
            P^.NEXT := ADD_SAME_POWERS (P^.NEXT)
       END;    { if }
      ADD_SAME_POWERS := P                               { מצביע לתחילת הרשימה }
   END;     { add_same_powers }
```

{ הפונקציה מקבלת שני מצביעים לשתי רשימות המייצגות פולינומים, }
{ ומחזירה מצביע לרשימה חדשה המכילה את תוצאות החיבור שלהם. }

```
FUNCTION ADD_POLYNOMS (P,
                       Q  : POLYNOM) : POLYNOM;
   VAR
      RESULT : POLYNOM;
   BEGIN
      CONNECT_POLYNOMS (P, Q, RESULT);                   { חיבור שתי רשימות }
      LIST_SORT (RESULT);                                { מיון רשימת התוצאה }
      ADD_POLYNOMS := ADD_SAME_POWERS (RESULT)           { החזרת מצביע לרשימה }
                                                         { לאחר צמצום של חזקות שוות. }
   END;      { add_polynoms }
```

```
   { הפונקציה מקבלת שני מצביעים לשתי רשימות המייצגות פולינומים, }
   { ומחזירה מצביע אל רשימה חדשה המכילה את המכפלה שלהם. }

FUNCTION MULTIPLY_POLYNOMS (P,
                            Q   : POLYNOM) : POLYNOM;
   VAR
      TEMP,
      PTR,
      RESULT : POLYNOM;
   BEGIN
      IF  (P = NIL) OR (Q = NIL) THEN           { אם אחת הרשימות ריקה }
         RESULT := NIL                          { התוצאה היא רשימה ריקה }
      ELSE
       BEGIN
         NEW (RESULT);                          { מצביע לראש רשימת התוצאה }
         PTR := RESULT;
         WHILE  P <> NIL  DO
          BEGIN              { לולאה על כל איברי הרשימה ש-P מצביע אליהם }
            TEMP := Q;
            WHILE  TEMP <> NIL  DO
             BEGIN           { לולאה על כל איברי הרשימה ש-Q מצביע עליהם }
               PTR^.FACTOR := P^.FACTOR * TEMP^.FACTOR; {הכפלת מקדמים}
               PTR^.POWER  := P^.POWER + TEMP^.POWER;  { חיבור חזקות }
               NEW (PTR^.NEXT);          { שירשור לאיבר הבא ברשימת התוצאה }
               PTR  := PTR^.NEXT;             { קידום מצביע ברשימת התוצאה }
               TEMP := TEMP^.NEXT                         { קידום מצביע Q }
             END;
            P := P^.NEXT                                  { קידום מצביע P }
          END;     { while p }
         PTR^.NEXT := NIL                         { ציון סוף רשימת התוצאה }
      END;   { else }
     LIST_SORT (RESULT);                               { מיון רשימת התוצאה }
     MULTIPLY_POLYNOMS := ADD_SAME_POWERS (RESULT)     { צמצום חזקות }
                                                   { שוות ברשימת התוצאה. }
   END;     { multiply_polynoms }

BEGIN  { main }
   P := READ_POLYNOM;                               { קריאת פולינום ראשון }
   Q := READ_POLYNOM;                               { קריאת פולינום שני }
   WRITELN;
   WRITELN (' :הסכום הוא':40);
   PRINT_POLYNOM (ADD_POLYNOMS (P, Q));          { הדפסת סכום הפולינומים }
   WRITELN (' :המכפלה היא':40);
   PRINT_POLYNOM (MULTIPLY_POLYNOMS (P, Q))     { הדפסת מכפלת הפולינומים }
END.
```

הסבר:

1. הפונקציה READ_POLYNOM היא פונקציה רקורסיבית, אשר קוראת זוגות מספרים שמייצגים מקדם ומעריך חזקה של איבר בפולינום. הפונקציה יוצרת רשימה מקושרת של איברי הפולינום ומחזירה מצביע לראש הרשימה. נתאר את

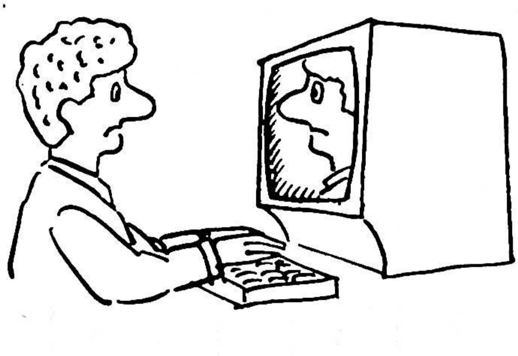
שלבי הפעולה שלה:

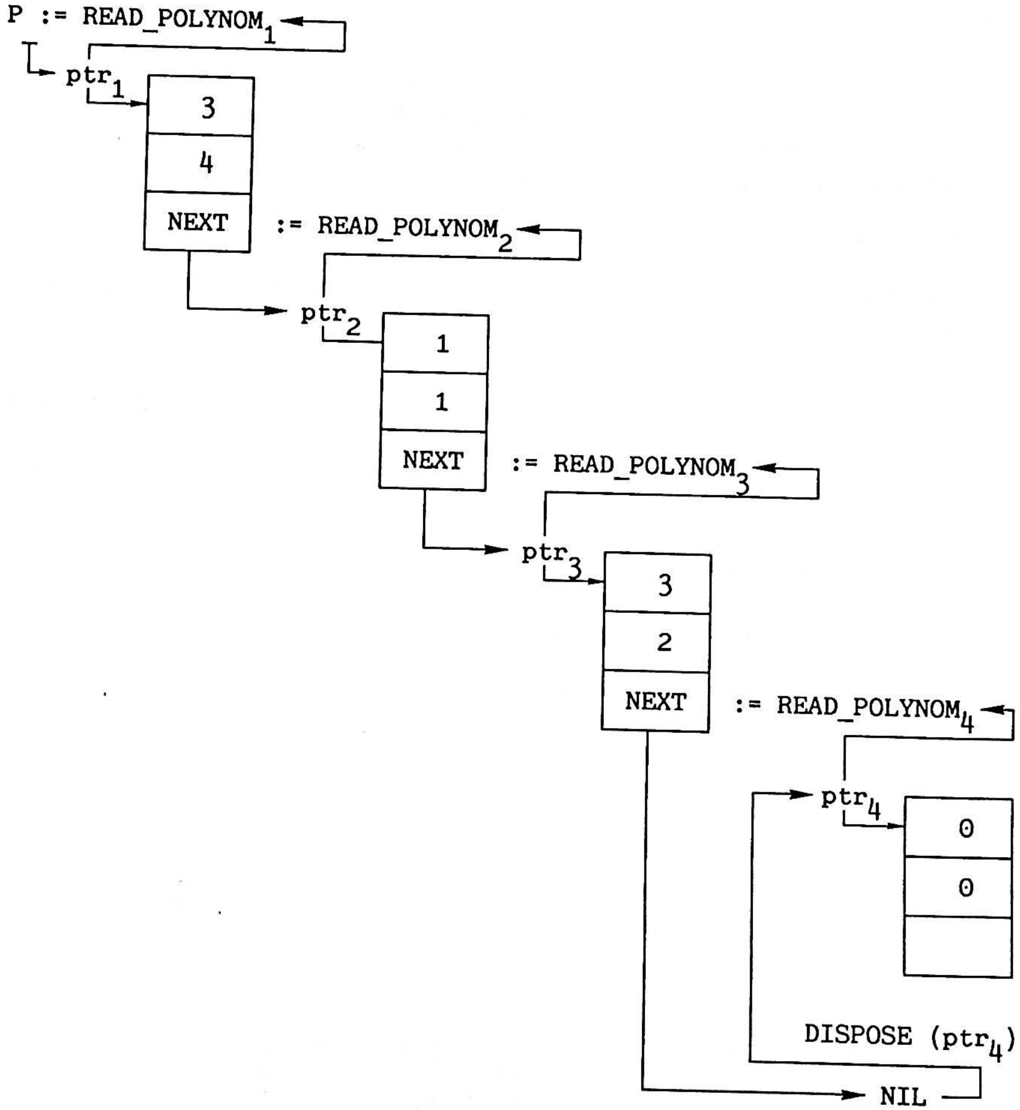
א. הקצאת מקום לרשומה החדשה.
ב. קריאת נתונים לתוך שדות הרשומה.
ג. אם שדה המקדם ברשומה הוא 0 (אפס), כלומר סוף הקלט, משחררים את המקום שהוקצה בזיכרון ומחזירים ערך NIL לשדה NEXT של הרשומה הקודמת.
אחרת, השדה NEXT של הרשומה הנוכחית יקבל את ערך המצביע המוחזר מקריאה רקורסיבית של READ_POLYNOM עבור שארית הקלט.
ד. הערך המוחזר ע"י הפונקציה הוא המצביע לרשומה הראשונה שנקראה.

נציג את התהליך הזה בתרשים:

נתונים הערכים: 4 3 המייצגים את הפולינום: $3x^4 + x + 3x^2$

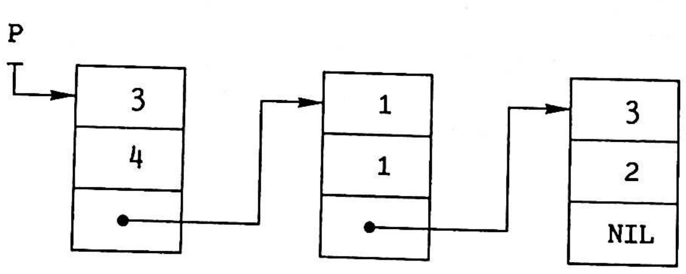
4	3
1	1
2	3
0	0





P := READ_POLYNOM1
ptr1
3
4
NEXT
:= READ_POLYNOM2
ptr2
1
1
NEXT
:= READ_POLYNOM3
ptr3
3
2
NEXT
:= READ_POLYNOM4
ptr4
0
0
DISPOSE (ptr4)
NIL

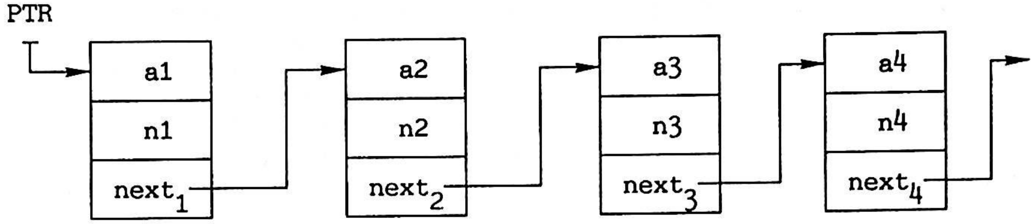
ומתקבל:



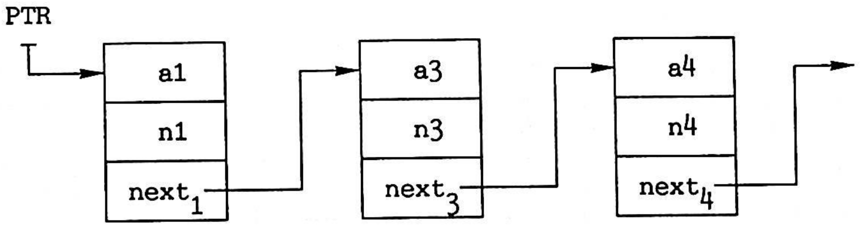
P
3
4
1
1
3
2
NIL

2. הפרוצדורה DELETE_PAIR מקבלת מצביע לרשומה מסוג PAIR, מוחקת את הרשומה הזו ומעדכנת את המצביע לרשומה הבאה. נתאר זאת בתרשים:

המצב הראשוני:



במהלך התוכנית הפרוצדורה מעדכנת ישירות את השדה NEXT של הרשומה הקודמת ברשימה. לאחר ביצוע הקריאה לפרוצדורה (PTR^.NEXT) DELETE_PAIR, נקבל את התוצאה הזו:



שים לב: המשתנה PTR^.NEXT מועבר לפרוצדורה כפרמטר משתנה (by reference).

3. הפרוצדורה CONNECT_POLYNOMS מקבלת שני מצביעים לראשי רשימות המייצגות פולינומים. היא מחזירה מצביע לרשימה <u>חדשה</u>, שהיא צירוף של שתי הרשימות האלו (אחת אחרי חברתה). היא עושה זאת בדרך הבאה:

<u>אם</u> שתי המחרוזות ריקות מחזירים NIL.
<u>אחרת:</u> - מעתיקים את הרשימה הראשונה לרשימה החדשה.
- מעתיקים את הרשימה השניה לרשימה החדשה, בהמשך למה שקיים בה.

4. הפונקציה ADD_SAME_POWERS מקבלת מצביע לרשימה <u>ממויינת</u> של רשומות מסוג PAIR. היא מחברת את שדות המקדמים של הרשומות שבהן שדות החזקה שווים, ומבטלת רשומות ששדה המקדם שלהן הוא אפס (0). הפונקציה מחזירה מצביע לרשימת התוצאה.

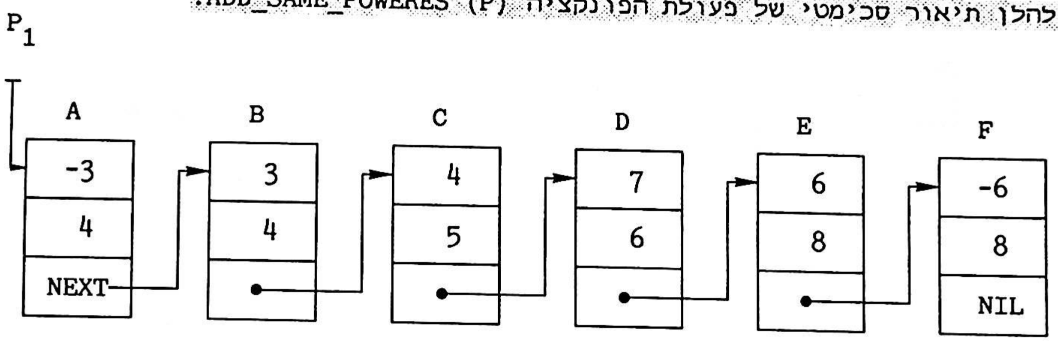
הפונקציה פועלת בדרך זו:

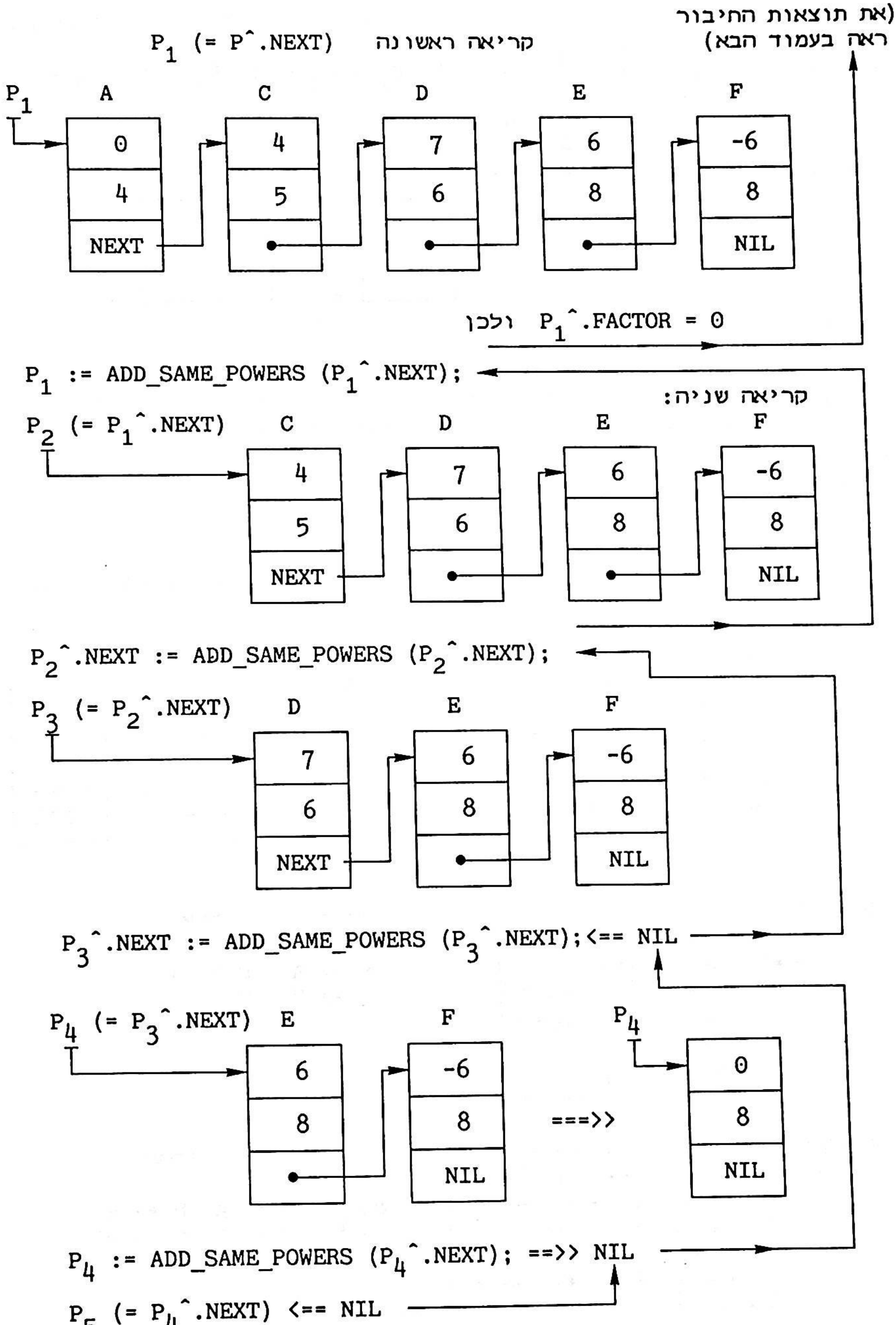
<u>אם</u> סוף הרשימה - מחזירים NIL.
<u>אחרת:</u> כל זמן שיש לפחות 2 רשומות ושדה החזקה שלהן שווה:
- מחברים מקדם של הרשומה השניה למקדם של הרשומה הראשונה.
- מוחקים את הרשומה השניה מהרשימה.

<u>אם</u> המקדם 0 (אפס):
- מוחקים את הרשומה.
- מחזירים את המצביע, שמתקבל מהקריאה הרקורסיבית לפונקציה עם פרמטר שהוא המצביע על הרשומה הבאה.

<u>אחרת:</u> השדה NEXT של הרשומה מקבל את המצביע החוזר מהקריאה הרקורסיבית לפונקציה.

להלן תיאור סכימטי של פעולת הפונקציה (P) ADD_SAME_POWERES:

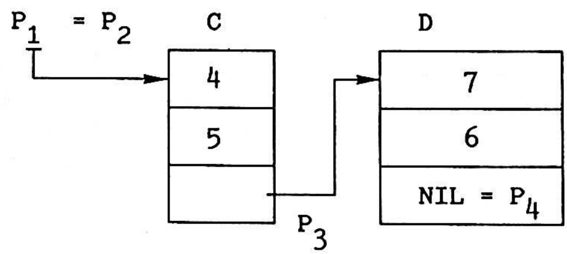




(את תוצאות החיבור
ראה בעמוד הבא)
P_1 (= P^.NEXT) קריאה ראשונה
P_1
A C D E F
0 4 7 6 -6
4 5 6 8 8
NEXT NIL
ולכן P_1^.FACTOR = 0
P_1 := ADD_SAME_POWERS (P_1^.NEXT);
קריאה שניה:
P_2 (= P_1^.NEXT)
C D E F
4 7 6 -6
5 6 8 8
NEXT NIL
P_2^.NEXT := ADD_SAME_POWERS (P_2^.NEXT);
P_3 (= P_2^.NEXT)
D E F
7 6 -6
6 8 8
NEXT NIL
P_3^.NEXT := ADD_SAME_POWERS (P_3^.NEXT);<== NIL
P_4 (= P_3^.NEXT)
E F
6 -6
8 8
NIL
===>>
P_4
0
8
NIL
P_4 := ADD_SAME_POWERS (P_4^.NEXT); ==>> NIL
P_5 (= P_4^.NEXT) <== NIL

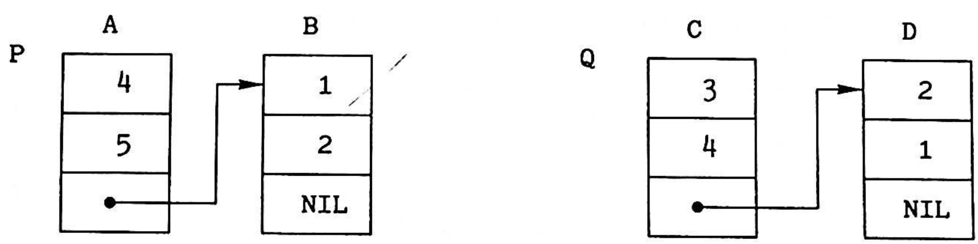
בסוף התהליך מקבלים:

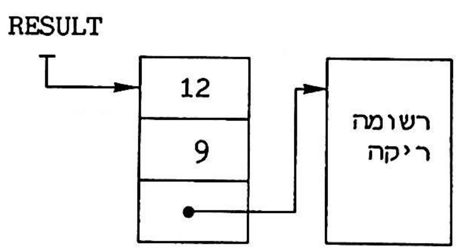
$P_1 = P_2$ ===>> P_3 ===>> P_4 = NIL



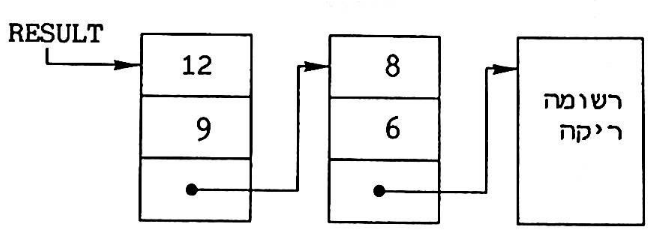
5. הפונקציה MULTIPLY_POLYNOMS מבצעת כפל של פולינומים. היא משתמשת בפרוצדורה למיון התוצאות והצגתו.

דוגמה סכימטית לפעולת הפונקציה MULTIPLY_POLINOMS (P, Q):

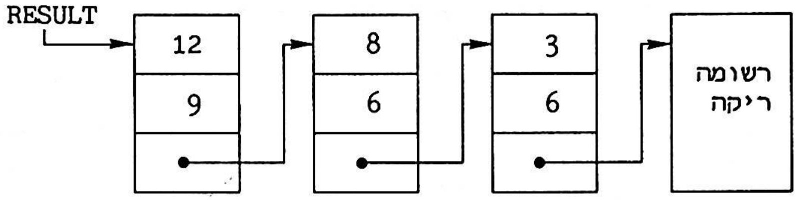




לפני	אחרי
P => A	P => A
Q => C	Q => D

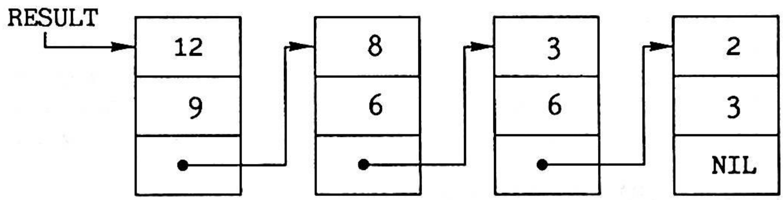


לפני	אחרי
P => A	P => B
Q => D	Q => NIL



לפני
P => B
Q => C

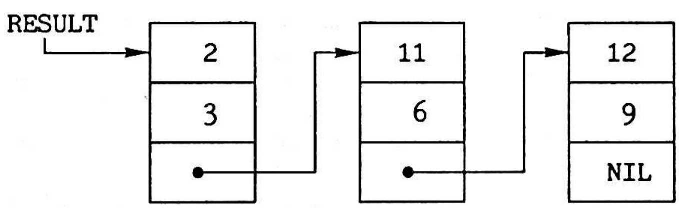
אחרי
P => B
Q => D



לפני
P => B
Q => D

אחרי
P => NIL
Q => NIL

תוצאה סופית תתקבל לאחר המיון וצמצום החזקות השוות:



תוכנית 4 (ראה תרגיל תכנות 4 בפרק 16)

```
PROGRAM TELEPHONE (DATA, OUTPUT);
CONST
   MAX_ROOM = 50;                         { מספר חדרים (טלפונים) מכסימלי }
TYPE
   PERSON_NAME = PACKED ARRAY [1..14] OF CHAR;
   TL_NUMBER   = PACKED ARRAY [1..10] OF '0'..'9';
   NEXT_PERSON = ^PERSON;
   PERSON = RECORD                        {                    רשומת משתמש }
               NAME : PERSON_NAME;        {                     שם משתמש   }
               NEXT : NEXT_PERSON         {           מצביע למשתמש הבא   }
            END;
   NUMBER = RECORD                        {                    רשומת טלפון }
               TL   : TL_NUMBER;          {                   מספר טלפון   }
               USER : NEXT_PERSON         {   מצביע למשתמש בראש הרשימה   }
            END;
   NUMBERS = ARRAY [1..MAX_ROOM] OF NUMBER;
                      { מערך של מספרי טלפון ומצביעים לרשימות המשתמשים }
VAR
   TL_LIST : NUMBERS;
   DATA    : TEXT;                        { קובץ הנתונים המכיל שמות ומספרי טלפון }
   LENGTH, I : INTEGER;

            { הפרוצדורה קוראת מקובץ הקלט את רשימת השמות ומספרי הטלפון.
              היא מחזירה מערך ממויין של מספרי טלפון ומצביעים לרשימה של
              שמות המשתמשים הממויינת בסדר אלפביתי הפוך, מ-'ת' ועד 'א'. }

PROCEDURE READ_DATA (VAR LIST : NUMBERS;
                     VAR LAST : INTEGER);
   VAR
      PTR    : NEXT_PERSON;
      P_NAME : PERSON_NAME;
      P_TL   : TL_NUMBER;
      I, J : INTEGER;

                                   { הפרוצדורה מוסיפה שם של משתמש חדש
                                     בראש רשימה, אשר PTR מצביע עליה. }

   PROCEDURE  ADD_NAME (VAR PTR    : NEXT_PERSON;        {    מצביע }
                            P_NAME : PERSON_NAME);       {   שם חדש }
      VAR
         TEMP : NEXT_PERSON;
      BEGIN
         NEW (TEMP);
         TEMP^.NEXT := PTR;
         TEMP^.NAME := P_NAME;
         PTR := TEMP
      END;   { add_name }
```

```
{ הפונקציה NUMBER_FOUND עורכת חיפוש בינרי של מספר הטלפון המבוקש
  ברשימת הטלפונים. אם הוא נמצא - היא מחזירה את מיקומו במשתנה MID
  ומקבלת ערך TRUE. אם המספר לא נמצא - הפונקציה מקבלת ערך FALSE,
  והמשתנה MID מצביע על המקום שבו המספר צריך להיות. }

FUNCTION NUMBER_FOUND (NUM    : TL_NUMBER; { מספר הטלפון המבוקש }
                       LAST   : INTEGER;   { מספר הרשומות
                                             במערך מספרי הטלפונים }
                   VAR MID    : INTEGER    ) : BOOLEAN;
   VAR
      FIRST : INTEGER;
   BEGIN
      FIRST := 1;
      REPEAT
         MID := (FIRST + LAST) DIV 2;
         IF  NUM < LIST [MID].TL THEN
            LAST := MID - 1
         ELSE
            FIRST := MID + 1
      UNTIL  (NUM = LIST [MID].TL) OR (FIRST > LAST);
      NUMBER_FOUND := NUM = LIST [MID].TL
   END; { number_found }

BEGIN  { read_data }
   LAST := 1;
   ASSIGN (DATA, 'DATA1');                    { פתיחת קובץ הנתונים }
   RESET (DATA);
   IF  NOT EOF (DATA)  THEN
      WITH  LIST [1]  DO
       BEGIN
         READLN (DATA, TL, P_NAME);  { קריאת ראשונה של מספר טלפון
         USER := NIL;                  ושם משתמש. }
         ADD_NAME (USER, P_NAME)               { הוספת השם לרשימה }
       END;
   WHILE  NOT EOF (DATA)  DO                  { לולאה עד סוף הקובץ }
    BEGIN
      READLN (DATA, P_TL, P_NAME);        { קריאה של מספר טלפון ושם }
      IF  NUMBER_FOUND (P_TL, LAST, I)  THEN     { אם המספר נמצא
        ברשימת הטלפונים יש להוסיף את שם המשתמש לרשימת
        המשתמשים של אותו מספר טלפון (מכיוון שהרשימה הנסרקת
        ממויינת בסדר הפוך, נקבל גם את הרשימה הזו באותו סדר מיון). }
         ADD_NAME (LIST [I].USER, P_NAME)
      ELSE          { אם המספר לא נמצא - זהו מספר טלפון חדש }
       BEGIN
         { יש לשבץ את מספר הטלפון החדש במקומו ברשימת הטלפונים.
           הפונקציה NUMBER_FOUND מחזירה את האינדקס של המערך שבו
           מופסק החיפוש. מספר הטלפון החדש צריך להיות לפני או אחרי
           מספר הטלפון שאת מקומו הסידורי מייצג MID.
           שים לב ש-MID חוזר במשתנה I. }
         IF  P_TL > LIST [I].TL  THEN           { מציאת המיקום המדויק }
            I := I + 1;
```

```
          FOR  J := LAST  DOWNTO  I  DO           {   הזזה - פינוי מקום }
             LIST [J + 1] := LIST [J];            { .למספר הטלפון החדש   }
          WITH  LIST [I]  DO
           BEGIN
             TL := P_TL;                          {   שיבוץ המספר למקומו }
             USER := NIL;
             ADD_NAME (USER, P_NAME)              {      הוספת השם לרשימה }
           END;
          LAST := LAST + 1                    { נוסף מספר לרשימת הטלפונים }
        END
      END;   { while not eof }
     CLOSE (DATA)
  END;  { read_data }

       { .פרוצדורה זו מדפיסה את מספרי הטלפון ובצידם את שמות המשתמשים-
                        .המשתנה LAST מייצג את מספר הטלפונים שברשימה }

PROCEDURE PRINT_DATA (VAR LIST : NUMBERS;
                          LAST : INTEGER);
   VAR
      I : INTEGER;

       { פרוצדורה זו מקבלת מצביע לראש רשימה של משתמשים (במספר טלפון
       .(מסוים) ומדפיסה את הרשימה מהסוף עד להתחלה (שים לב לרקורסיה
        השמות מודפסים לפי סדר אלפביתי, כי סדר המיון בו נבנתה רשימת
                   .'השמות היה סדר יורד, כלומר מ-'ת' ועד 'א
                 .כתוצאה - בשחרור הרקורסיה, השמות מודפסים בסדר עולה }

   PROCEDURE PRINT_LIST (PTR : NEXT_PERSON);
   BEGIN
      IF  PTR^.NEXT <> NIL  THEN
         PRINT_LIST (PTR^.NEXT);
      WRITE (PTR^.NAME:14)
   END;    { print_list }

   BEGIN { print_data }
      WRITELN (' מספר טלפון                  שמות משתמשים');
      WRITELN ('------------------------------------------------');
      FOR  I := 1  TO  LAST  DO
         WITH  LIST [I]  DO
          BEGIN
            WRITE (TL:10);                          { הדפסת מספר טלפון }
            PRINT_LIST (USER);     { הדפסת רשימת המשתמשים באותו טלפון }
            WRITELN
          END
   END;  { print_data }
```

```
BEGIN  { main }
   READ_DATA (TL_LIST, LENGTH);
   PRINT_DATA (TL_LIST, LENGTH)
END.   { main }
```

הסבר:

נבנה מערך של 50 רשומות מסוג NUMBER, אשר כוללות שדה של מספר טלפון (TL) ושדה שבו יימצא מצביע לתחתלת רשימת המשתמשים. הקלט לתוכנית זו הוא קובץ אשר מכיל שם משתמש ומספר טלפון. הקובץ ממויין לפי שם משתמש בסדר אלפביתי.

כאשר נקרא את שם המשתמש ומספר טלפון מהרשימה, יתכנו 2 המקרים הבאים:

א. מספר הטלפון איננו מופיע ברשימת הטלפונים: יש להוסיף את המספר במקום המתאים במיון על ידי שיבוץ, ולשרשר אליו רשומה מסוג PERSON שתכיל את שם המשתמש.

ב. מספר הטלפון מופיע ברשימת הטלפונים: נוסיף את שם המשתמש לתחילת הרשימה של המשתמשים. רשימות המשתמשים בכל מספר טלפון יהיו ממויינות בסדר אלפביתי יורד.

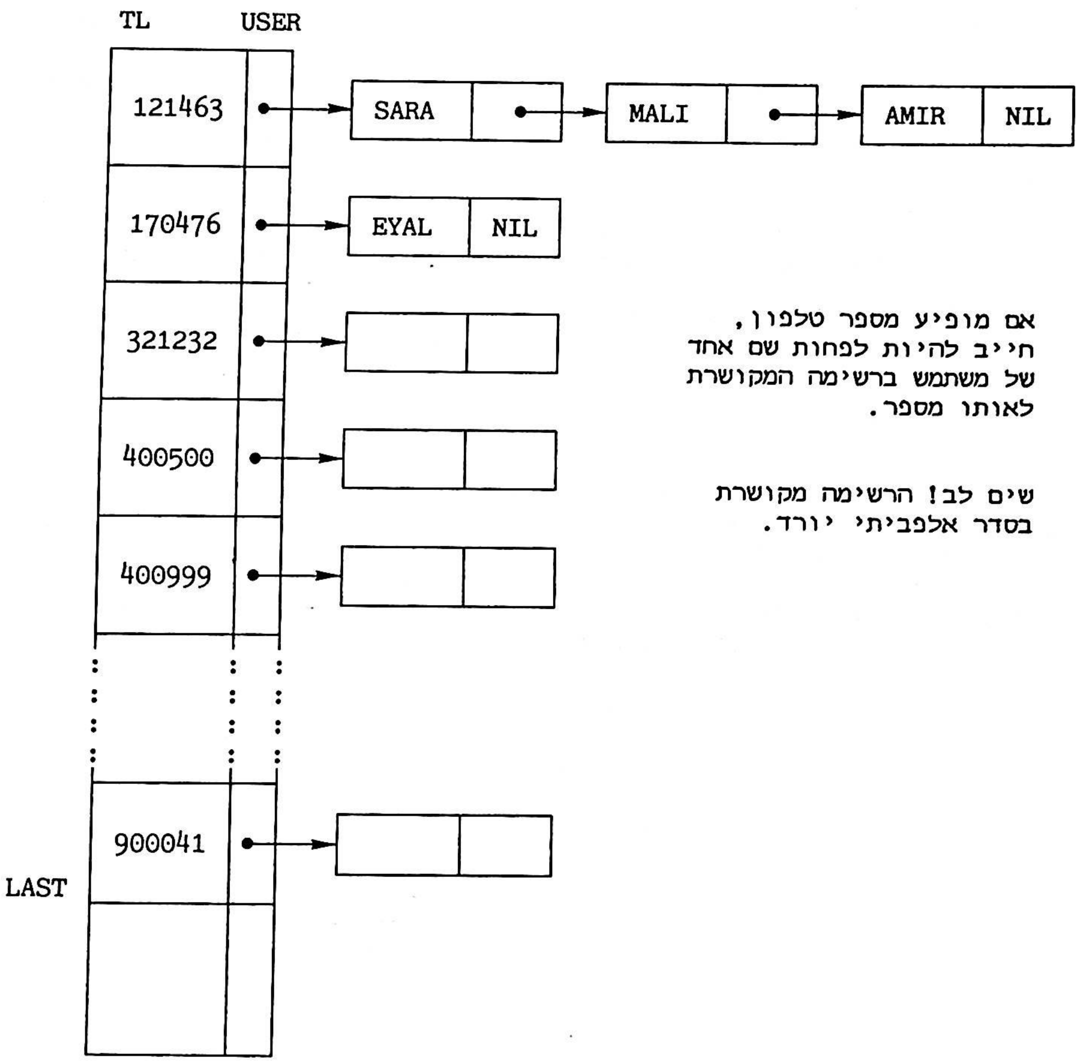
שיקולים לבחירת מבנה הנתונים:

חלופה לשימוש במערך, יכולה להיות רשימה מקושרת של רשומות מסוג NUMBER. מכיוון שנתון שמספר הרשומות (= מספר הטלפונים) הוא 50, לא היינו מקבלים חיסכון במקום. החיפוש ברשימה מקושרת אחר המקום להכנסת המספר בכל שלב היה חייב להיות חיפוש לינארי ולא בינרי כפי שניתן לבצע על המערך.

החיסכון היחידי בשימוש ברשימה מקושרת הוא, שבשלב השיבוץ אין צורך בהזזה של שארית המערך ממקום השיבוץ ועד הרשומה האחרונה.

נותר מצב ביניים כלשהו בתוכנית, ונציג שני מקרים של הוספת משתמש.

נתון המצב הבא:



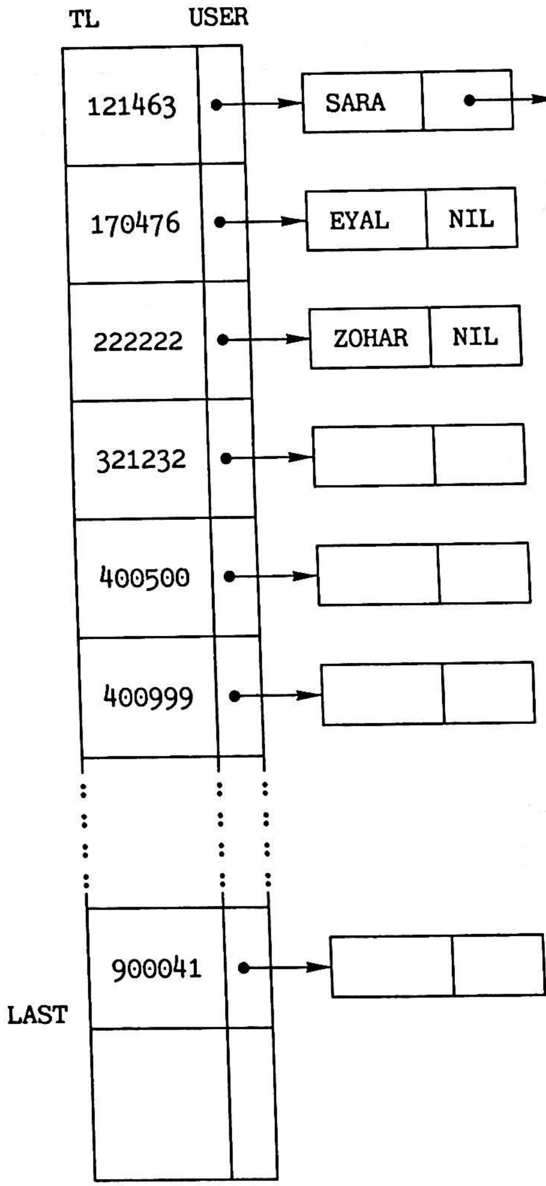
אם מופיע מספר טלפון,
חייב להיות לפחות שם אחד
של משתמש ברשימה המקושרת
לאותו מספר.

שים לב! הרשימה מקושרת
בסדר אלפביתי יורד.

מקרה ראשון (בהתייחס לדוגמה):

מוסיפים שם משתמש ומספר טלפון, אשר אינו נמצא ברשימה:
נתונים: מספר טלפון - 222222
שם משתמש - ZOHAR

לאחר החוספה נקבל:

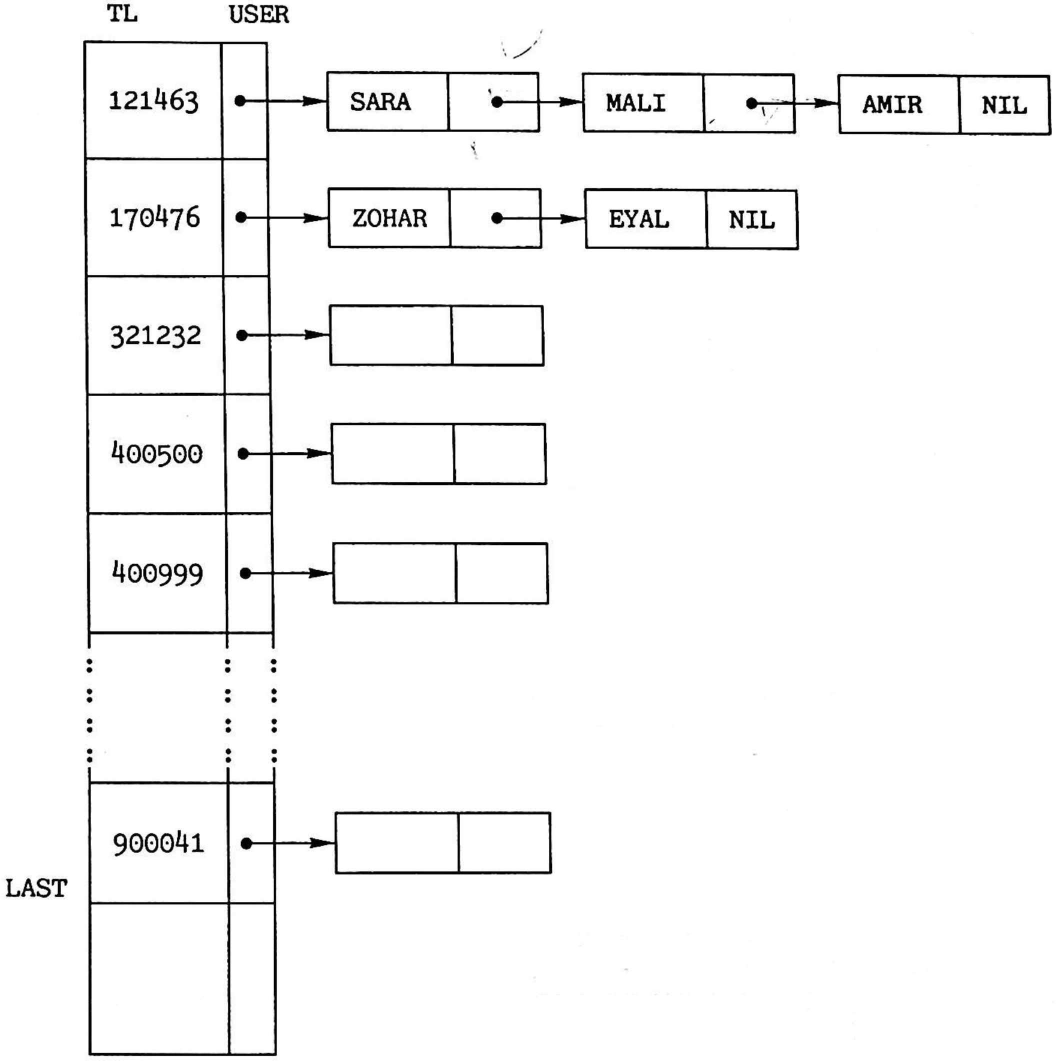


הערך של LAST גדל באחד.

מקרה שני (בהתייחס לדוגמה):

הוספת שם משתמש למספר הטלפון, אשר נמצא ברשימה:
נתונים: מספר טלפון - 170476
שם משתמש - ZOHAR

לאחר ההוספה נקבל:



תוכנית 6 (ראה תרגיל תכנות 6 בפרק 16)

```
PROGRAM STACK_POINTERS (INPUT, OUTPUT);
CONST
   END_OF_INPUT = -9999;
TYPE
   NEXT_NUMBER = ^NUMBER;
   NUMBER = RECORD
               N    : INTEGER;
               NEXT : NEXT_NUMBER
            END;
VAR
   STACK : NEXT_NUMBER;
   NUM   : INTEGER;

PROCEDURE MAKE_NULL (VAR STACK : NEXT_NUMBER);        { איפוס המחסנית }
   BEGIN
      STACK := NIL
   END;    { make_null }

                                        { הכנסת הנתון NUM לראש המחסנית }

PROCEDURE PUSH (VAR  STACK : NEXT_NUMBER;
                     NUM   : INTEGER);
   VAR
      PTR : NEXT_NUMBER;
   BEGIN
      NEW (PTR);                                     { הקצאת מקום בזיכרון }
      PTR^.N    := NUM;             { שדה N ברשומה יקבל את הערך של NUM }
      PTR^.NEXT := STACK;          { שדה NEXT של הרשומה החדשה יצביע על }
                                   {      הרשומה הנמצאת בראש המחסנית. }
      STACK := PTR                   {        עדכון המצביע לראש המחסנית. }
                                     { עכשיו הרשומה החדשה היא הראשונה }
   END;    { push }

                    { הפונקציה מחזירה ערך TRUE כאשר המחסנית STACK ריקה }

FUNCTION END_OF_STACK (STACK : NEXT_NUMBER ) : BOOLEAN;
   BEGIN
      END_OF_STACK := STACK = NIL              { NIL מסמן את סוף המחסנית }
   END;    { end_of_stack }
```

```
{ פונקציה זו מחזירה את הערך שבראש המחסנית STACK }

FUNCTION  POP (VAR STACK : NEXT_NUMBER) : INTEGER;
   VAR
      TEMP : NEXT_NUMBER;
   BEGIN
     TEMP  := STACK;                 { מצביע לראש המחסנית }
     STACK := STACK^.NEXT;  { עדכון מצביע ראש המחסנית שיצביע לאיבר הבא }
     POP   := TEMP^.N;               { הערך בראש המחסנית }
     DISPOSE (TEMP)          { שיחרור הרשומה שהיתה קודם בראש המחסנית }
   END;     { pop }

BEGIN  { main }
   MAKE_NULL (STACK);
   READLN (NUM);
   WHILE  NUM <> END_OF_INPUT  DO            { קריאה עד סימן סוף קלט }
    BEGIN
      PUSH (STACK, NUM);                     { הכנסת איבר למחסנית }
      READLN (NUM)
    END;
   WHILE  NOT END_OF_STACK (STACK)  DO       { לולאה עד סוף המחסנית }
      WRITELN (POP (STACK))                  { הדפסת איברי המחסנית }
END.
```

<u>דוגמה:</u>

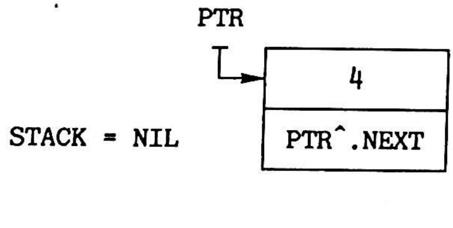
הקלט הוא: 4
1
7
-9999

יצירת מחסנית ריקה ע"י הפקודה: MAKE_NULL (STACK).

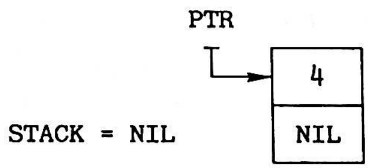
המצביע לראש המחסנית: STACK = NIL.

נציג את השלבים להכנסת הערך 4 למחסנית:

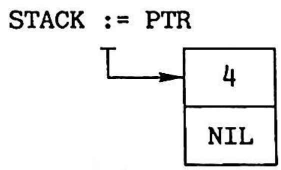
א. ביצוע הפקודה NEW(PTR). בסיומה PTR יצביע לרשומה חדשה בזכרון.
הכנסת הערך 4 לשדה N.



ב. הכנסת המצביע לראש המחסנית, אל השדה NEXT של האיבר החדש.
כלומר, הכנסת ערך המצביע STACK (במקרה זה שווה ל-NIL) לשדה
PTR^.NEXT. ונקבל עתה:

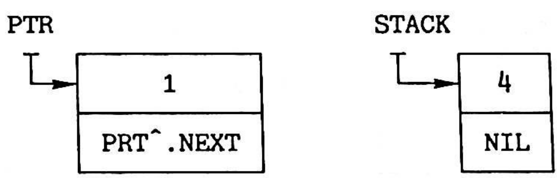


ג. עדכון המצביע STACK כך, שיצביע לראש המחסנית החדשה.
כלומר STACK מקבל את ערכו של PTR. נקבל:

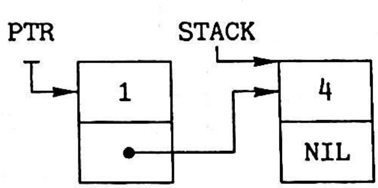


נוסיף עתה את הערך 1 למחסנית:

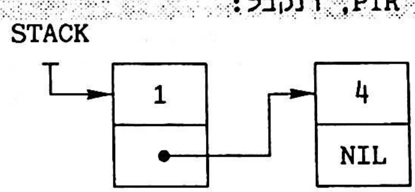
א. ביצוע הפקודה NEW(PTR). בסיומה המצביע PTR יצביע לרשומה חדשה
בזכרון. הכנסת הערך 1 לשדה N.



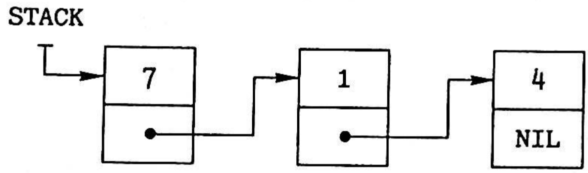
ב. PTR^.NEXT מקבל את ערכו של המצביע STACK.



ג. עדכון המצביע STACK כך, שיצביע לראש המחסנית.
כלומר STACK מקבל את ערכו של PTR, ונקבל:



באותה צורה לאחר הוספת האיבר הבא בקלט, נקבל:

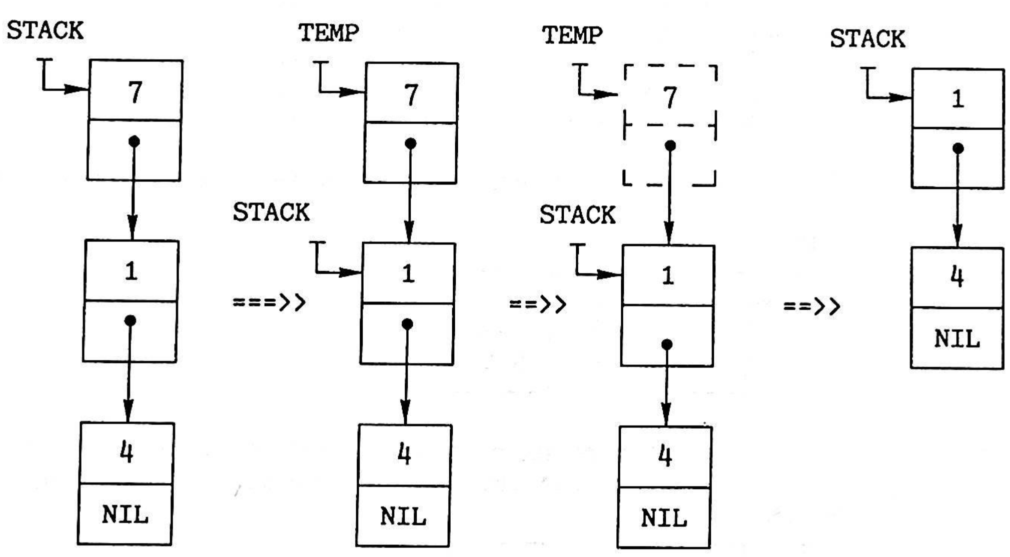


תהליך ההוצאה של איבר מהמחסנית: POP (STACK)

1. שמירת המצביע למחסנית ב-TEMP, ועדכון המצביע לראש המחסנית.
STACK:=STACK^.NEXT

2. שחרור המקום ש-TEMP מצביע עליו.

נקבל:



```
PROGRAM SAS (DATA, OUTPUT);
CONST
   MID_OF_STRING = 'A';
TYPE
   NEXT_CHARACTER = ^CHARACTER;
   CHARACTER = RECORD
                    CH   : CHAR;
                    NEXT : NEXT_CHARACTER
                END;
VAR
   STACK : NEXT_CHARACTER;
   CH    : CHAR;
   EQUAL : BOOLEAN;
   DATA  : TEXT;

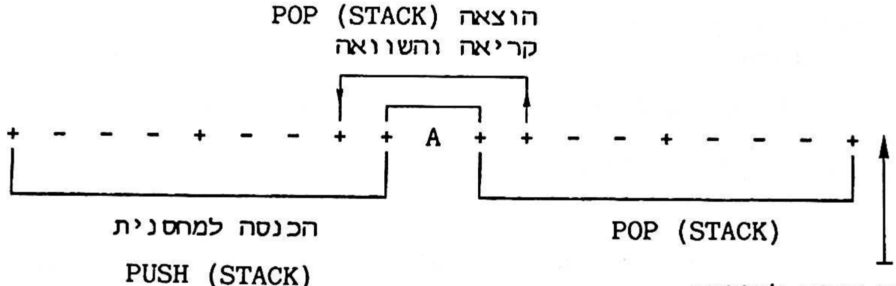
PROCEDURE MAKE_NULL (VAR STACK : NEXT_CHARACTER);
   BEGIN
      STACK := NIL
   END;   { make_null }

PROCEDURE PUSH (VAR  STACK : NEXT_CHARACTER;
                     C     : CHAR);
   VAR
      PTR : NEXT_CHARACTER;
   BEGIN
      NEW (PTR);
      PTR^.NEXT := STACK;
      STACK     := PTR;
      STACK^.CH := C
   END;   { push }

FUNCTION END_OF_STACK (STACK : NEXT_CHARACTER ) : BOOLEAN;
   BEGIN
      END_OF_STACK := STACK = NIL
   END;   { end_of_stack }

FUNCTION  POP (VAR STACK : NEXT_CHARACTER) : CHAR;
   VAR
      PTR : NEXT_CHARACTER;
   BEGIN
      PTR   := STACK;
      POP   := STACK^.CH;
      STACK := STACK^.NEXT;
      DISPOSE (PTR)
   END;    { pop }
```

```
BEGIN  { main }
   EQUAL := TRUE;
   ASSIGN (DATA, 'DATA');                               { פתיחת קובץ הנתונים }
   RESET (DATA);
   MAKE_NULL (STACK);                                   { איפוס המחסנית }
   IF  NOT EOF (DATA)  THEN
      READ (DATA, CH);                                  { קריאת התו הראשון }
   WHILE  (CH <> MID_OF_STRING) AND NOT EOF (DATA)  DO
    BEGIN     { לולאה לקריאת תווים עד התו 'A', המוגדר ב-MID_OF_STRING }
                                        { ומציין את אמצע רשימת התווים. }
      PUSH (STACK, CH);                                 { הכנסה של תו למחסנית }
      READ (DATA, CH)                                   { קריאת של תו נוסף. }
    END;
   WHILE  NOT (END_OF_STACK (STACK) OR EOF (DATA))  DO
    BEGIN
      READ (DATA, CH);                                  { קריאת של תו נוסף }
      IF  (CH <> POP (STACK)) AND NOT (CH IN ['+', '-'])  THEN
         EQUAL := FALSE
         { השוואת התו האחרון שנקרא, עם תמונת הראי של המחסנית (STACK).
                       תמונת הראי נבנית ע"י הורקה של המחסנית. }
    END;
   EQUAL := EQUAL AND END_OF_STACK (STACK) AND EOF (DATA);
   WRITELN (EQUAL)                         { TRUE - אם המחרוזת מקיימת את התנאים }
END.
```



בשלב זה צריך להיות:

א. סוף המחסנית לאחר תהליך ההורקה.

ב. סוף הקלט.

תוכנית 8 (ראה תרגיל תכנות 8 בפרק 16)

```
TYPE                                            { הגדרת סוגי משתנים }
   ELEMENT_TYPE = CHAR;
   NEXT_ELEMENT = ^ELEMENT_REC;
   ELEMENT_REC  = RECORD
                     ELEMENT : ELEMENT_TYPE;
                     NEXT    : NEXT_ELEMENT
                  END;

                                  {      הפרוצדורות MAKE_NULL, PUSH
                                         והפונקציות EMPTY ו-POP
                                  הוסברו בתוכניות קודמות בפרק זה. }

PROCEDURE MAKE_NULL (VAR STACK : NEXT_ELEMENT);
   BEGIN
      STACK := NIL
   END;     { make_null }

PROCEDURE PUSH (VAR  STACK : NEXT_ELEMENT;
                     C     : ELEMENT_TYPE);
   VAR
      PTR : NEXT_ELEMENT;
   BEGIN
      NEW (PTR);
      PTR^.NEXT       := STACK;
      STACK           := PTR;
      STACK^.ELEMENT  := C
   END;     { push }

FUNCTION EMPTY (STACK : NEXT_ELEMENT ) : BOOLEAN;
   BEGIN
      EMPTY := STACK = NIL
   END;     { empty }

FUNCTION  POP (VAR STACK : NEXT_ELEMENT) : ELEMENT_TYPE;
   VAR
      TEMP : NEXT_ELEMENT;
   BEGIN
      TEMP  := STACK;
      POP   := STACK^.ELEMENT;
      STACK := STACK^.NEXT;
      DISPOSE (TEMP)
   END;    { pop }
```

{ הפרוצדורה המבוקשת }

```
PROCEDURE HACHLAFA (VAR SRC_STACK   : NEXT_ELEMENT;    { מחסנית מקור }
                    VAR NATUN       : ELEMENT_TYPE;    { נתון במחסנית }
                        MISPAR_EVAR : INTEGER);        { מספר בנתון }
  VAR
     DEST_STACK : NEXT_ELEMENT;
     I : INTEGER;
  BEGIN
     MAKE_NULL (DEST_STACK);                           { איפוס מחסנית המטרה }
     WHILE  NOT EMPTY (SRC_STACK)  DO                  { העברת האיברים ממחסנית }
        PUSH (DEST_STACK, POP (SRC_STACK)); { המקור למחסנית המטרה. }
     I := 0;
     WHILE  NOT EMPTY (DEST_STACK)  DO
      BEGIN                                 { העברה ממחסנית המטרה למחסנית המקור }
        I := I + 1;                                    { תוך כדי ספירת איברים. }
        IF  I = MISPAR_EVAR  THEN                      { מציאת האיבר המבוקש }
         BEGIN
           NATUN := POP (DEST_STACK);                  { האיבר המבוקש נמצא }
           PUSH (SRC_STACK, NATUN)                     { החזרתו למחסנית המקור }
         END
        ELSE
           PUSH (SRC_STACK, POP (DEST_STACK))          { החזרת שאר האיברים }
                                                       { למחסנית המקור. }
      END;
     IF  I < MISPAR_EVAR  THEN                         { מספר האיבר גדול }
                                                       { ממספר האיברים במחסנית. }
        WRITELN ('MISPAR EVAR IS OVER STACK SIZE! ')
     ELSE
        WRITELN (NATUN)
  END;   { hachlafa }
```

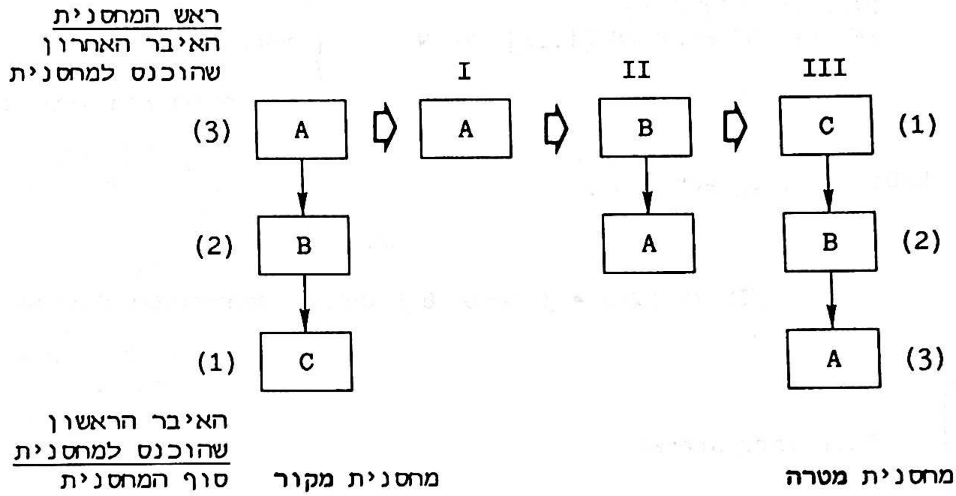
<u>הסבר:</u>

המחסנית צריכה להישאר ללא שינוי לאחר האיתור של האיבר המבוקש. מספר האיברים במחסנית לא ידוע, לכן יש צורך להגיע לאיבר בתחתית המחסנית ולחזור לכיוון האיבר העליון תוך כדי ספירתם. התהליך מתבצע ע"י העברת תוכן המחסנית למחסנית עזר באופן, שהאיבר מראש מחסנית המקור, יוכנס בכל שלב לראש מחסנית המטרה. לאחר פעולה זו מתקבל סדר הפוך של האיברים במחסנית המטרה.

כעת נחזיר באותו אופן את האיברים ממחסנית המטרה למחסנית המקור, ותוך כדי כך נספור את האיברים לפי סדר הכנסתם. בצורה זו נמצא את האיבר שאת מיקומו אנו מחפשים.

כאשר מספרו הסידורי של האיבר שמחפשים גדול ממספר האיברים במחסנית, נזהה זאת על ידי קבלת סוף המחסנית לפני שמונה האיברים יגיע למספר המבוקש.

תרשים סכימטי של תהליך יצירת מחסנית המטרה.



פתרון נוסף

אפשר לפתור את הבעיה גם בדרך רקורסיבית.
הרעיון הוא לבצע "צלילה" לקרקעית המחסנית ולספור את האיברים תוך כדי "ציפה" למעלה. גם כאן, אין משנים את מחסנית המקור.

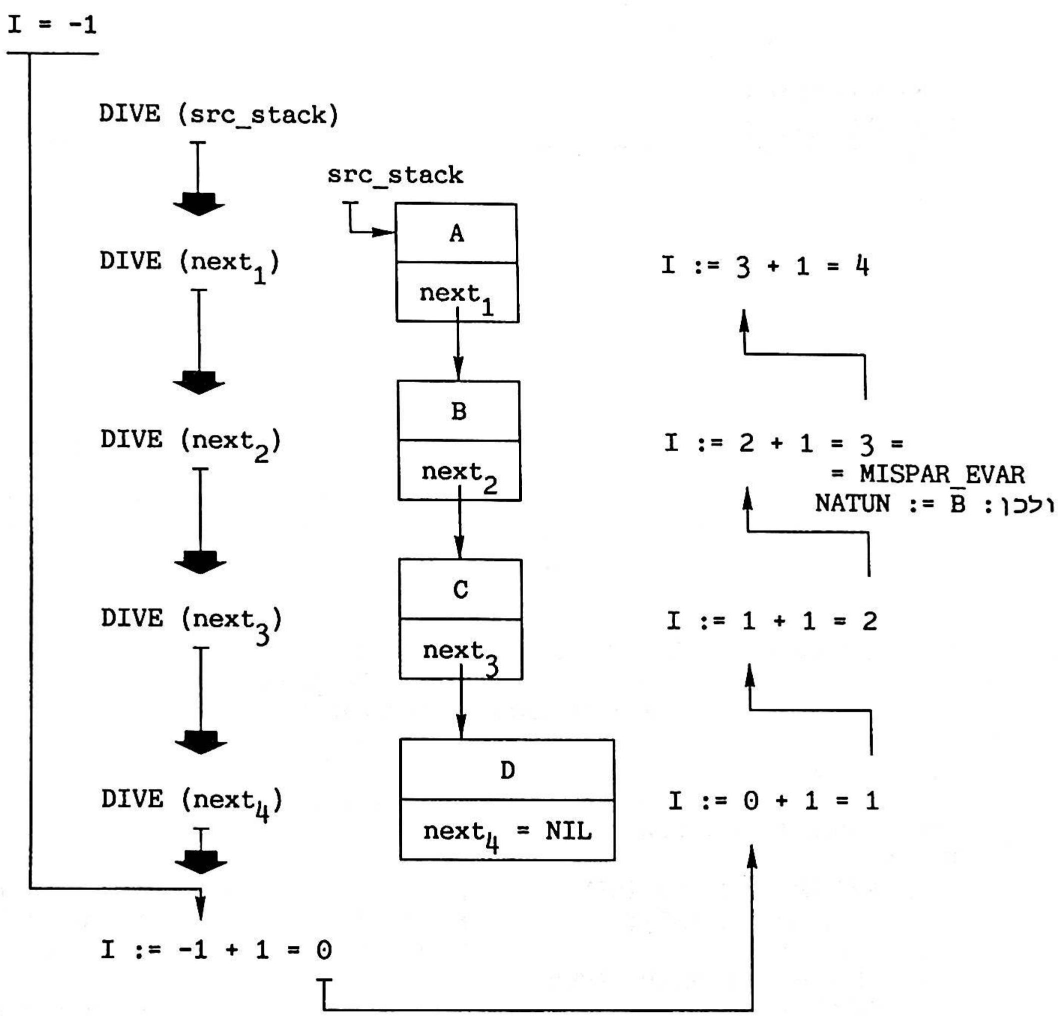
```
PROCEDURE B_HACHLAFA  (VAR SRC_STACK   : NEXT_ELEMENT;
                        VAR NATUN       : ELEMENT_TYPE;
                            MISPAR_EVAR : INTEGER);

   VAR
      I : INTEGER;

   PROCEDURE DIVE (PTR : NEXT_ELEMENT);
      BEGIN
        IF NOT EMPTY (PTR) THEN         { אם לא סוף המחסנית -            }
           DIVE (PTR^.NEXT);            { קריאה רקורסיבית לפרוצ' DIVE }
        I := I + 1;                     { ספירת האיברים מסוף המחסנית  }
        IF  I = MISPAR_EVAR  THEN
           NATUN := PTR^.ELEMENT                          { האיבר המבוקש }
      END;    { dive }
```

```
BEGIN     { b_hachlafa }
   I := -1;
   DIVE (SRC_STACK);
   IF  MISPAR_EVAR IN [1..I]  THEN        { בשלב זה, I מכיל את מספר }
                                          {          האיברים במחסנית. }
      WRITELN (NATUN)                       { האיבר בתחום המחסנית }
   ELSE
      WRITELN ('MISPAR_EVAR IS OVER STACK SIZE! ')
END;      { b_hachlafa }
```

נתאר את מהלך הפרוצדורה B_HACHLAFA, כאשר MISPAR_EVAR = 3



תוכנית 9 (ראה תרגיל תכנות 9 בפרק 16)

```
PROGRAM SWS (DATA, OUTPUT);
CONST
   MID_OF_S = 'A';
   MID_OF_W = 'B';
TYPE
   NEXT_CHARACTER = ^CHARACTER;
   CHARACTER = RECORD
                  CH   : CHAR;
                  NEXT : NEXT_CHARACTER
               END;
VAR
   STACK_S,
   STACK_W  : NEXT_CHARACTER;
   CH       : CHAR;
   EQUAL    : BOOLEAN;
   DATA     : TEXT;

PROCEDURE MAKE_NULL (VAR STACK : NEXT_CHARACTER);
   BEGIN
      STACK := NIL
   END;   { make_null }

PROCEDURE PUSH (VAR  STACK : NEXT_CHARACTER;
                     C     : CHAR);
   VAR
      PTR : NEXT_CHARACTER;
   BEGIN
      NEW (PTR);
      PTR^.NEXT := STACK;
      STACK     := PTR;
      STACK^.CH := C
   END;   { push }

FUNCTION END_OF_STACK (STACK : NEXT_CHARACTER ) : BOOLEAN;
   BEGIN
      END_OF_STACK := STACK = NIL
   END;    { end_of_stack }

FUNCTION  POP (VAR STACK : NEXT_CHARACTER) : CHAR;
   VAR
      TEMP : NEXT_CHARACTER;
   BEGIN
      TEMP  := STACK;
      POP   := STACK^.CH;
      STACK := STACK^.NEXT;
      DISPOSE (TEMP);
   END;   { pop }
```

```
BEGIN { main }
   EQUAL := TRUE;
   ASSIGN (DATA, 'DATA');                        { פתיחת קובץ נתונים }
   RESET (DATA);
   MAKE_NULL (STACK_S);                          { איפוס מחסנית }
   MAKE_NULL (STACK_W);
   IF  NOT EOF (DATA)  THEN
      READ (DATA, CH);
   WHILE  (CH <> MID_OF_S) AND NOT EOF (DATA)  DO
    BEGIN                    { STACK_W-ו STACK_S קריאת תווים לתוך }
                             {   .MID_OF_S עד התו 'A' המוגדר ע"י  }
      PUSH (STACK_S, CH);
      PUSH (STACK_W, CH);
      READ (DATA, CH)
    END;
   WHILE  NOT (END_OF_STACK (STACK_S) OR EOF (DATA))  DO
    BEGIN              { המשך קריאת תווים והכנסתם ל-STACK_W תוך כדי
                       {       .STACK_S השוואתם לתמונת הראי של מחסנית
                       .STACK_S תמונת הראי נבנית ע"י הורקת מחסנית }
      PUSH (STACK_W, CH);
      READ (DATA, CH);
      IF  (CH <> POP (STACK_S)) AND NOT (CH IN ['+', '-'])  THEN
         EQUAL := FALSE
    END;
   WHILE  (CH <> MID_OF_W) AND NOT EOF (DATA)  DO
    BEGIN                { ,(MID_OF_W) 'B' המשך קריאת תווים עד התו }
                         {                 .STACK_W והכנסתם למחסנית }
      PUSH (STACK_W, CH);
      READ (DATA, CH)
    END;
   WHILE  NOT (END_OF_STACK (STACK_W) OR EOF (DATA)) AND EQUAL DO
    BEGIN            {        קריאת התווים לאחר אמצע המחרוזת תוך כדי
                        השוואתם לתמונת הראי. תמונת הראי היא הפעם  של
                        .מחסנית STACK_W. היא נבנית ע"י הורקת המחסנית }
      READ (DATA, CH);
      IF  (CH <> POP (STACK_W)) AND NOT (CH IN ['/', '*'])  THEN
         EQUAL := FALSE
    END;
   EQUAL := EQUAL AND END_OF_STACK (STACK_W) AND EOF (DATA);
   WRITELN (EQUAL)
END.
```

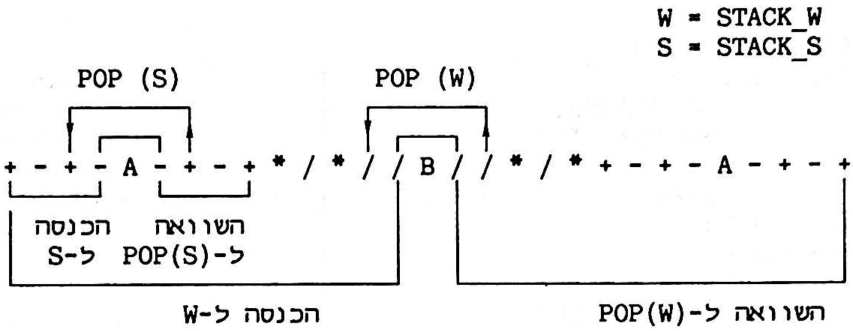
הסבר:

הפתרון של תוכנית זו דומה לפתרון תוכנית 7 בפרק זה.
בפתרון זה מופיעים פרוצדרות ופונקציות שפותחו והוסברו בתוכניות אחרות.

בתרגיל זה ישנן שתי מחסניות ויש צורך לזהות מחרוזת מהצורה הבאה:

$s\ A\ s^t\ u\ B\ u^t\ (s\ A\ s^t)^t$

במחסנית אחת (S) משתמשים כדי לבדוק את חוקיות המחרוזת: $s\ A\ s^t$ (ראה תוכנית 7).
במחסנית השניה (W) נשתמש באותה דרך כדי לבדוק עד התו 'B'.





תוכנית 10 (ראה תרגיל תכנות 10 בפרק 16)

```
PROGRAM COMPUTER_QUEUE (DATA, OUTPUT);
CONST
   RUN_TIME = 2;
TYPE
   JOB_PTR = ^JOB_REC;
   JOB_REC = RECORD                                  { רשומת תוכנית }
                JOB_NUMBER,                          { מספר תוכנית }
                SERVICE_TIME,                        { זמן שירות }
                TIME            : INTEGER;           { זמן מופע }
                NEXT            : JOB_PTR            { מצביע לעבודה הבאה בתור }
             END;
   QUEUE = RECORD                                    { רשומת תור }
              FIRST,                                 { מצביע לאיבר הראשון }
              LAST   : JOB_PTR                       { מצביע לאיבר אחרון }
           END;
VAR
   DATA      : TEXT;                                 { קובץ נתונים }
   PRIORITY  : 1..2;                                 { עדיפות }
   Q         : ARRAY [1..2] OF QUEUE;                { מערך של שני תורים }
   CPU_TIME,
   T, I                 : INTEGER;
   RUN_JOB         : JOB_PTR;                        { התוכנית שכרגע מורצת }
   NEW_JOB         : JOB_REC;                        { תוכנית שמגיעה לתור }
   MORE            : BOOLEAN;        { משתנה המורה של-NEW_JOB נקראה תוכנית
                                       שזמן המופע שלה גדול מ-CPU_TIME,
                                       על כן היא נמצאת בהמתנה. }
   NEW_PRIORITY : 1..2;                              { עדיפות }

FUNCTION MIN (A, B  : INTEGER) : INTEGER;
   BEGIN
      IF  A < B  THEN
         MIN := A
      ELSE
         MIN := B
   END;    { min }

PROCEDURE MAKE_NULL (VAR Q : QUEUE);                 { יצירת תור ריק }
   BEGIN
      WITH  Q  DO
       BEGIN
         NEW (FIRST);
         LAST := FIRST                { המצביע על האיבר האחרון שווה
                                        למצביע על האיבר הראשון. }
       END
   END;    { make_null }
```

```
{ הפונקציה EMPTY מחזירה TRUE כאשר התור ריק.            }
{ כלומר המצביעים לאיבר הראשון ולאיבר האחרון - שווים. }

FUNCTION EMPTY (VAR Q : QUEUE) : BOOLEAN;
   BEGIN
      EMPTY := Q.FIRST = Q.LAST
   END;    { empty }

{ פרוצדורה זו מוסיפה לסוף התור Q את רשומת התוכנית JOB }

PROCEDURE ADD_TO_QUEUE (VAR Q   : QUEUE;
                            JOB : JOB_REC);
   BEGIN
      WITH  Q  DO
       BEGIN
         LAST^  := JOB;
         NEW (LAST^.NEXT);
         LAST := LAST^.NEXT
       END
   END;     { add_to_queue }

{ פרוצדורה זו מבטלת את רשומת התוכנית שנמצאת בראש תור Q }

PROCEDURE REMOVE_FROM_QUEUE (VAR Q : QUEUE);
   VAR
      TEMP : JOB_PTR;
   BEGIN
      IF  NOT  EMPTY (Q)  THEN
         WITH  Q  DO
          BEGIN
            TEMP  := FIRST;
            FIRST := FIRST^.NEXT;
            DISPOSE (TEMP)
          END
   END;     { remove_from_queue }

{ פונקציה זו מחזירה את המצביע לרשומת התוכנית שבראש תור Q }

FUNCTION  FRONT (VAR Q : QUEUE) : JOB_PTR;
   BEGIN
      FRONT := Q.FIRST
   END;     { front }

BEGIN    { main }
   MAKE_NULL (Q [1]);                          { איפוס שני התורים }
   MAKE_NULL (Q [2]);
   ASSIGN (DATA, 'DATA');                      { פתיחת קובץ        }
   RESET (DATA);
   IF  NOT EOF (DATA)  THEN
     WITH  NEW_JOB  DO        { קריאת נתוני התוכנית הראשונה ועדיפות }
       READLN (DATA, JOB_NUMBER, SERVICE_TIME, TIME, NEW_PRIORITY);
```

```
CPU_TIME := NEW_JOB.TIME;                { זמן הגעה של התוכנית הראשונה }
MORE := TRUE;
WRITELN ('מספר עבודה   יעד דקה', CPU_TIME:5,
        'זמן התחלת הרצה של תוכנית ראשונה: ');
REPEAT
   WHILE  NOT EOF (DATA) AND (NEW_JOB.TIME <= CPU_TIME) DO
    BEGIN         { זמן הגעה של תוכנית חדשה קטן או שווה לזמן הנוכחי }
     MORE := TRUE;
     ADD_TO_QUEUE (Q [NEW_PRIORITY], NEW_JOB);          { הוספה של }
                                   { תוכנית חדשה לתור, לפי עדיפות. }
     WITH  NEW_JOB  DO
      READLN                       { קריאה של תוכנית נוספת ועדיפות. }
       (DATA, JOB_NUMBER, SERVICE_TIME, TIME, NEW_PRIORITY)
    END;
   IF  (NEW_JOB.TIME <= CPU_TIME) AND MORE  THEN
    BEGIN
                                   { ישנה תוכנית שממתינה להכנס לתור }
     MORE := FALSE;     { אין רשומות שנקראו ומחכות לזמן מופע מתאים }
     ADD_TO_QUEUE (Q [NEW_PRIORITY], NEW_JOB)            { התוכנית }
              { האחרונה שנקראה מצטרפת לתור לפי סדר העדיפות שלה. }
    END;
  PRIORITY := 2;                   { העדיפות לביצוע ניתנת לתור מספר 2 }
  T := CPU_TIME;                   { זמן CPU לפני הרצת התוכנית }
  WHILE  (PRIORITY >= 1) AND (T = CPU_TIME)  DO  { לולאה המתבצעת
                                    כל זמן שהעדיפות גדולה מ-0 }
                                    { ועדיין לא רצה כל תוכנית שהיא. }
    IF  NOT  EMPTY (Q [PRIORITY])  THEN            { התור אינו ריק }
     BEGIN
      RUN_JOB := FRONT (Q [PRIORITY]);      { התוכנית להרצה }
                                            { נמצאת בראש התור. }
       WITH  RUN_JOB^  DO
        BEGIN
         FOR  I := 1  TO  MIN (RUN_TIME, SERVICE_TIME)  DO
           זמן הרצת התוכנית יהיה הזמן המינימלי בין זמן השירות }
           { לבין הזמן המכסימלי להרצה, כפי שנקבע מראש.
          WRITELN (JOB_NUMBER:7, CPU_TIME + I:10);
                   { הדפסת זמן הרצה עכשווי ומספר התוכנית שבהרצה }
        CPU_TIME := CPU_TIME + MIN (RUN_TIME, SERVICE_TIME);
                                            { קידום זמן CPU }
        SERVICE_TIME := SERVICE_TIME -
                        MIN (RUN_TIME, SERVICE_TIME);
                               עדכון זמן השירות של התוכנית, }
                               { שהוא משך הזמן הנותר לסיומה.
        IF  SERVICE_TIME > 0  THEN           { עדיין לא הסתיים }
                                              { זמן השירות. }
         ADD_TO_QUEUE (Q [PRIORITY], RUN_JOB^);       { הוספת }
                                            { התוכנית לסוף התור. }
     END;   { with run_job }
    REMOVE_FROM_QUEUE (Q [PRIORITY])
                 הסרת התוכנית מראש התור. אפשריים שני מצבים: }
                 - התוכנית הסתיימה.
                 { - התוכנית הועברה לסוף התור עם יתרת זמן השירות שלה.
```

```
        END
       ELSE                                          { התור ריק. }
        PRIORITY := PRIORITY - 1;                    { העדיפות ניתנת לתור 1. }
    IF  T = CPU_TIME  THEN                           { אם לא הורצה תוכנית }
     BEGIN
       WRITELN ('WAITING', CPU_TIME:10);
       CPU_TIME := CPU_TIME + 1            { ה-CPU ממתין יחידת זמן אחת }
     END
  UNTIL                                    { הלולאה מתבצעת כל עוד: }
      EOF (DATA)    AND                    { - קריאת הקלט לא הסתיימה }
      EMPTY (Q [1]) AND                    { - ישנם תוכניות להרצה בתור 1 }
      EMPTY (Q [2]) AND                    { - ישנם תוכניות להרצה בתור 2 }
      NOT MORE                { - ישנה תוכנית הממתינה בגלל זמן המופע שלה }
END.
```

<u>הסבר:</u>

העבודות בקלט מסודרות לפי זמני המופע במערכת.
כאשר קוראים את נתוני התוכנית ל-NEW_JOB ומגלים שזמן המופע (הגעה) שלה גדול מזמן CPU בפועל (CPU_TIME), אי אפשר להכניס אותה לתור וגם אי אפשר לסיים את ההרצה בכללותה. המשתנה MORE מציין זאת ע"י ערך TRUE.

התוכנית מכילה 2 תורים הנמצאים במערך בגודל 2. התור שנמצא במקום 1 הוא בעל עדיפות 1, והתור במקום 2 הוא של עדיפות 2 (ראה שרטוט מצורף).

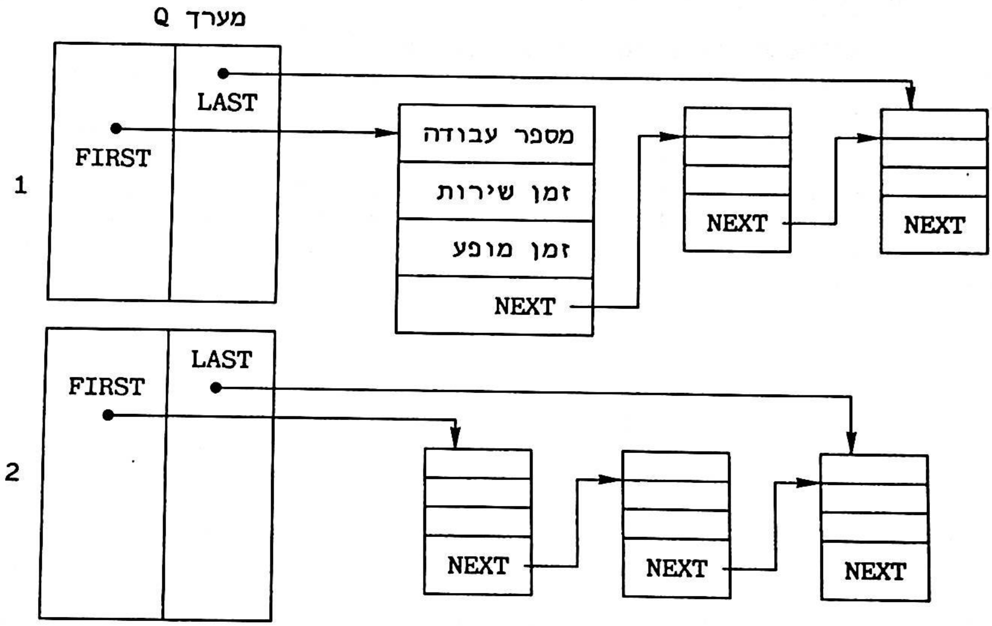
המשתנה CPU_TIME מורה על הזמן. ערכו ההתחלתי הינו זמן המופע של העבודה הראשונה. לאחר מכן מתבצעת לולאה שבה:
א. קוראים את נתוני כל העבודות שזמן המופע שלהם קטן או שווה ל-CPU_TIME ומכניסים אותן לתור המתאים לפי העדיפות שלהן.
ב. בוחרים עבודה מראש התור בעל העדיפות הגבוהה ביותר, שבו יש עבודות לביצוע ומריצים אותה למשך זמן, שהוא המינימום בין 2 דקות (זמן הרצה מכסימלי) לבין זמן השירות שנותר לעבודה.

אם אין עבודה הממתינה לביצוע באף לא אחד מהתורים, אך עדיין יש תוכנית בקלט, מקדמים את CPU_TIME ב-1 (כלומר, המחשב המתין יחידת זמן אחת) וחוזרים אל הלולאה.

לאחר ההרצה נעדכן את זמן השירות שנותר לעבודה. אם זמן השירות של התוכנית לפני ההרצה היה גדול מזמן ההרצה המכסימלי כפי שנקבע, נותר עדיין זמן שירות ונעדכן אותו עבור אותה תוכנית. נכניס את התוכנית הזו לסוף התור המתאים וניגש לביצוע התוכנית הבאה.

כאשר זמן השירות של התוכנית לפני ההרצה היה קטן מזמן השירות המכסימלי, המשמעות היא שהתוכנית הסתיימה. אפשר לעבור מיד לביצוע התוכנית הבאה.

תיאור של מבני הנתונים:



רשומה מסוג QUEUE

רשומות מסוג JOB_REC

תוכנית 12 (ראה תרגיל 12 בפרק 16)

```
PROGRAM DE_QUEUE (COMMANDS, OUTPUT);
TYPE
   NAME  = PACKED ARRAY [1..15] OF CHAR;
   Q_PTR = ^Q_ELEMENT;
   Q_ELEMENT = RECORD                            { רשומה בתור          }
                  ELEMENT : NAME;                { שם                  }
                  NEXT,                          { מצביע לאיבר הבא     }
                  BEFORE  : Q_PTR                { מצביע לאיבר הקודם   }
               END;
   DEQUEUE = RECORD                              { רשומת מצביעים       }
                HEAD,                            { מצביע לראש התור     }
                TAIL   : Q_PTR                   { מצביע לסוף התור     }
             END;
VAR
   COMMANDS : TEXT;                              { קובץ פקודות }
   DEQ      : DEQUEUE;
   COM1,
   COM2     : CHAR;
   NEW_NAME : NAME;

PROCEDURE MAKE_NULL (VAR Q : DEQUEUE);           { יצירה ואיפוס של התור }
   BEGIN
      WITH  Q  DO
       BEGIN
         NEW (HEAD);
         TAIL := HEAD       { המצביע לסוף התור שווה למצביע לראש התור }
       END
   END;    { make_null }

                                  { הפונקציה מחזירה ערך TRUE כאשר התור ריק }

FUNCTION EMPTY (Q : DEQUEUE) : BOOLEAN;
   BEGIN
      WITH  Q  DO
         EMPTY := HEAD = TAIL
   END;    { empty }
                                                 { הכנסת איבר לראש התור }

PROCEDURE INSERT_FRONT (VAR Q        : DEQUEUE;
                        VAR NEW_NAME : NAME);
   BEGIN
      WITH  Q  DO
       BEGIN
         HEAD^.ELEMENT := NEW_NAME;
         NEW (HEAD^.NEXT);
         HEAD^.NEXT^.BEFORE := HEAD;
         HEAD := HEAD^.NEXT
       END
   END;    { insert_front }
```

```
                                          { הכנסת איבר לסוף התור }
PROCEDURE INSERT_BACK (VAR Q        : DEQUEUE;
                       VAR NEW_NAME : NAME);
   BEGIN
      WITH  Q  DO
       BEGIN
         NEW (TAIL^.BEFORE);
         TAIL^.BEFORE^.NEXT := TAIL;
         TAIL := TAIL^.BEFORE;
         TAIL^.ELEMENT := NEW_NAME
       END
   END;     { insert_back }

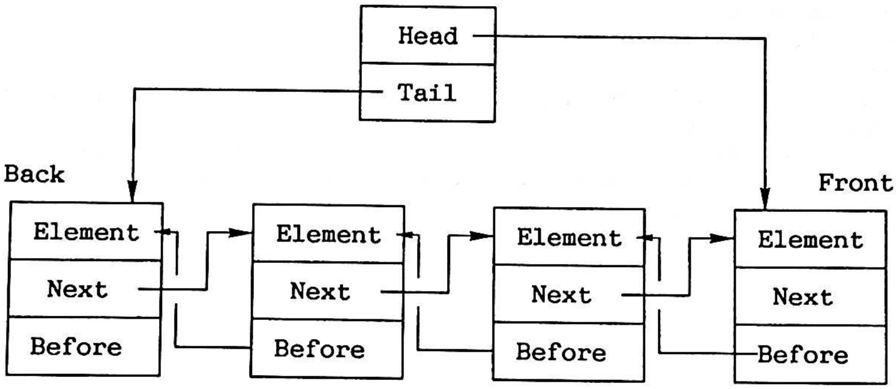
PROCEDURE REMOVE_FRONT (VAR Q : DEQUEUE);           { יציאה מראש התור }
   BEGIN
      IF  NOT  EMPTY (Q)  THEN
         WITH  Q  DO
          BEGIN
            HEAD := HEAD^.BEFORE;
            DISPOSE (HEAD^.NEXT)
          END
   END; { remove_front }

PROCEDURE REMOVE_BACK (VAR Q : DEQUEUE);            { יציאה מסוף התור }
   BEGIN
      IF  NOT EMPTY (Q)  THEN
         WITH  Q  DO
          BEGIN
            TAIL := TAIL^.NEXT;
            DISPOSE (TAIL^.BEFORE)
          END
   END; { remove_back }

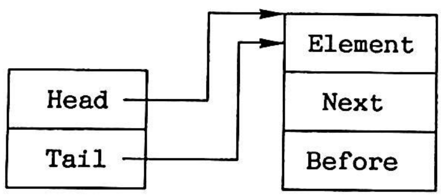
PROCEDURE PRINT_DEQ (VAR Q : DEQUEUE);                   { הדפסת התור }
   VAR
      TEMP : Q_PTR;
   BEGIN
      WRITELN ('השמות בתור הם:':30);
      WITH  Q  DO
       BEGIN
         TEMP := HEAD;
         WHILE  TEMP <> TAIL  DO
          BEGIN
            TEMP := TEMP^.BEFORE;
            WRITELN (TEMP^.ELEMENT)
          END  { while }
       END  { with }
   END;    { print_deq }
```

```
BEGIN  { main }
   MAKE_NULL (DEQ);                                              { איפוס התור }
   ASSIGN (COMMANDS, 'COMMANDS.DAT');
   RESET (COMMANDS);
   WHILE  NOT EOF (COMMANDS)  DO
    BEGIN
      READ (COMMANDS, COM1);                            { קריאת הנתון הראשון }
      CASE  COM1  OF
         'מ' : BEGIN                                  { קריאת הנתון השני והשם }
                 READLN (COMMANDS, COM2, NEW_NAME);
                 CASE  COM2  OF                             { case בתוך case }
                    'ר' : INSERT_FRONT (DEQ, NEW_NAME);
                                                              { הכנסה לראש התור }
                    'ס' : INSERT_BACK (DEQ, NEW_NAME)
                                                              { הכנסה לסוף התור }
                 END    { case com2 }
              END;
         'י' : BEGIN
                 READLN (COMMANDS, COM2);                { קריאת הנתון השני }
                 CASE  COM2  OF
                    'ר' : REMOVE_FRONT (DEQ);              { הוצאה מראש התור }
                    'ס' : REMOVE_BACK (DEQ)                { הוצאה מסוף התור }
                 END
              END;
         'ה' : BEGIN
                 READLN (COMMANDS);                   { מעבר לשורה הבאה בקלט }
                 PRINT_DEQ (DEQ)                      {             הדפסת התור }
              END
      END  { case com1 }
    END;   { while }
   CLOSE (COMMANDS)
END.
```

הרשומה ש-Head מצביע עליה, תמיד ריקה.



תור ריק:



תוכנית 13 (ראה תרגיל תכנות 13 בפרק 16)

```
PROGRAM TREES;
TYPE
   ELEMENT_TYPE = INTEGER;                        { דוגמא לסוג טיפוס }
   NODE_PTR = ^NODE;                              { מצביע לצומת בעץ }
   NODE = RECORD                                  { רשומת צומת בעץ }
             ELEMENT      : ELEMENT_TYPE;         { האיבר }
             LEFT_CHILD,                          { מצביע לבן שמאלי }
             RIGHT_CHILD : NODE_PTR               { מצביע לבן ימני }
          END;
   BIN_TREE = NODE_PTR;          { עץ בינרי מיוצג ע"י מצביע לצומת, }
                                 { שהוא שורש העץ. }
VAR
   TREE,
   TEMP  : BIN_TREE;

PROCEDURE MAKE_NULL (VAR TREE : BIN_TREE);        { יצירת עץ בינרי ריק }
   BEGIN
      TREE := NIL
   END;     { make_null }

          { פונקציה זו מחזירה TRUE כאשר האלמנט X מופיע בעץ TREE }

FUNCTION MEMBER (X    : ELEMENT_TYPE;
                 TREE : BIN_TREE)    : BOOLEAN;
   BEGIN
      IF  TREE = NIL  THEN
         MEMBER := FALSE
      ELSE                                        { העץ לא ריק }
         IF  TREE^.ELEMENT = X  THEN
            MEMBER := TRUE                        { X נמצא בצומת הנוכחי }
         ELSE
            IF  TREE^.ELEMENT > X  THEN  { אם האיבר הנוכחי גדול מ-X }
                                  { יש לבדוק אם X נמצא בתת-העץ השמאלי. }
               MEMBER := MEMBER (X, TREE^.LEFT_CHILD)
            ELSE                          { X גדול מהאיבר הנוכחי }
                                  { ויש לבדוק אם X נמצא בתת-העץ הימני. }
               MEMBER := MEMBER (X, TREE^.RIGHT_CHILD)
   END;     { member }
```

```
                                         { פרוצדורה להכנסת איבר X לעץ TREE }
PROCEDURE INSERT (X        : ELEMENT_TYPE;
                  VAR TREE : BIN_TREE);
  BEGIN
    IF  TREE = NIL  THEN
     BEGIN
       NEW (TREE);                                   { כאשר העץ ריק. }
       WITH  TREE^  DO                               { הוספה בצומת הנוכחי }
        BEGIN
          ELEMENT := X;
          LEFT_CHILD := NIL;
          RIGHT_CHILD := NIL
        END
     END
    ELSE
                                                     { העץ אינו ריק }
      { אם X קטן מהאלמנט בצומת הנוכחי - הכנס את X בתת-העץ השמאלי,
        אחרת: אם X גדול מהאלמנט בצומת הנוכחית - הכנס את X בתת-העץ הימני,
        אחרת: איבר X כבר נמצא בעץ ואין צורך להוסיפו.
        אין איברים כפולים.                                              }
      IF  TREE^.ELEMENT > X  THEN
        INSERT (X, TREE^.LEFT_CHILD)
      ELSE
        IF  TREE^.ELEMENT < X  THEN
          INSERT (X, TREE^.RIGHT_CHILD)
                        { המסקנה במקום זה היא ש-X נמצא בעץ ...else }
  END;   { insert }

                        { פונקציה זו מחזירה את הערך של האלמנט הקטן
                          ביותר בעץ TREE ומותקת את הצומת בה הוא נמצא. }

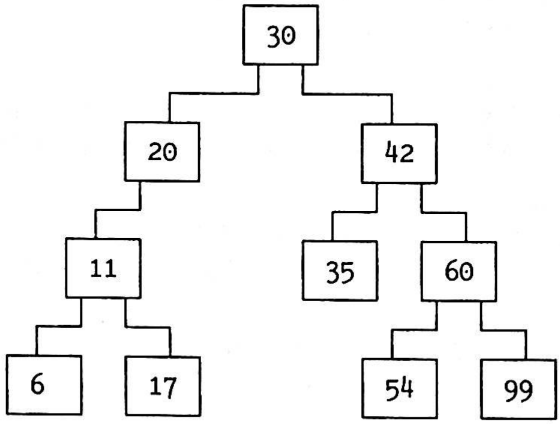
FUNCTION DELETE_MIN (VAR TREE : BIN_TREE) : ELEMENT_TYPE;
  BEGIN
    IF  TREE^.LEFT_CHILD = NIL  THEN
     BEGIN           { הצומת השמאלי ביותר בעץ הינו האיבר הקטן ביותר }
       DELETE_MIN := TREE^.ELEMENT;
       TEMP := TREE;                       { TEMP הוא משתנה עזר גלובלי }
       TREE := TREE^.RIGHT_CHILD;
       DISPOSE (TEMP)
     END
    ELSE
       DELETE_MIN := DELETE_MIN (TREE^.LEFT_CHILD) { תהליך רקורסיבי
                                 של חיפוש האיבר הקטן (השמאלי) ביותר. }
  END;   { delete_main }
```

{ פרוצדורה זו מוחקת את האלמנט X מהעץ TREE }

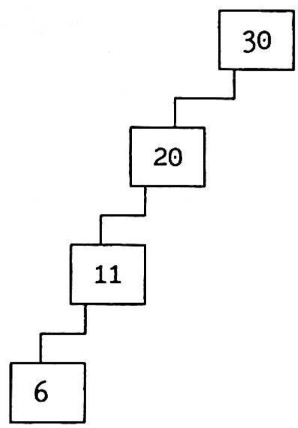
```
PROCEDURE DELETE (X    : ELEMENT_TYPE;
           VAR TREE : BIN_TREE);
  BEGIN
    IF  TREE <> NIL  THEN                          { כאשר העץ לא ריק }
      IF  X < TREE^.ELEMENT  THEN          { אם X קטן מהאיבר הנוכחי }
        DELETE (X, TREE^.LEFT_CHILD)  { התקדם שמאלה בעץ בתהליך
                                                      רקורסיבי. }
      ELSE
      IF  X > TREE^.ELEMENT  THEN           { אם X גדול מהאיבר הנוכחי }
        DELETE (X, TREE^.RIGHT_CHILD)  { התקדם לימין העץ בתהליך
                                                      רקורסיבי. }
      ELSE                                { האיבר X נמצא בצומת הנוכחי }
        IF  (TREE^.LEFT_CHILD = NIL) AND          { איבר X הוא עלה }
                (TREE^.RIGHT_CHILD = NIL)  THEN
         BEGIN                                 { ללא בן ימני ובן שמאלי }
           DISPOSE (TREE);        { מחק מהעץ את הצומת שמכילה את X }
           TREE := NIL
         END
        ELSE              { איבר X אינו נמצא בעלה, יש לו בן אחד לפחות }
          IF  TREE^.LEFT_CHILD = NIL  THEN
           BEGIN      { לאיבר X אין בן שמאלי, אבל יש לו בן ימני }
             TEMP := TREE;  { קישור האיבר שהצביע על X  עם האיבר
             TREE := TREE^.RIGHT_CHILD;          ש-X מצביע עליו. }
             DISPOSE (TEMP)
           END
          ELSE
            IF  TREE^.RIGHT_CHILD = NIL  THEN
             BEGIN                                  { לאיבר X אין בן ימני }
               TEMP := TREE;
               TREE := TREE^.LEFT_CHILD;  { קישור האיבר שהצביע
                                    על X עם האיבר ש-X מצביע עליו. }
               DISPOSE (TEMP)
             END
            ELSE                   { לאיבר X יש בן ימני ובן שמאלי
                                 מכניסים במקום X את האיבר הקטן }
                                 ביותר מתת-העץ הימני. הסדר נשמר. }
               TREE^.ELEMENT := DELETE_MIN (TREE^.RIGHT_CHILD)
  END;   { delete }

PROCEDURE INORDER (TREE : BIN_TREE);          { הדפסת איברי העץ לפי סדר }
  BEGIN
    IF  TREE <> NIL  THEN
     BEGIN                                     { אם לא מגיעים לסוף העץ: }
       INORDER (TREE^.LEFT_CHILD);              { הדפסת תת-העץ השמאלי }
       WRITE (TREE^.ELEMENT:4);                  { הדפסת הצומת הנוכחי }
       INORDER (TREE^.RIGHT_CHILD)              { הדפסת תת-העץ הימני }
     END
  END;   { inorder }
```

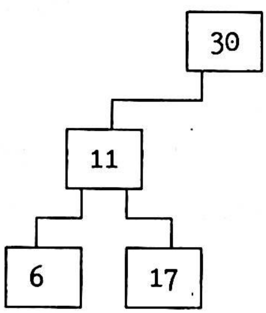
נתון העץ הבא:



מחיקת עלה, למשל 17.
תת-העץ השמאלי יראה כך:

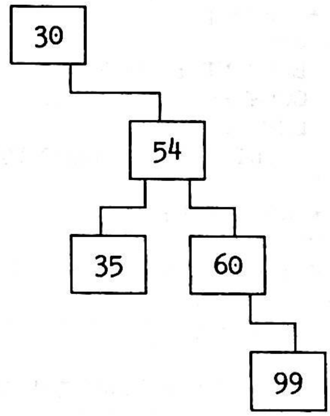


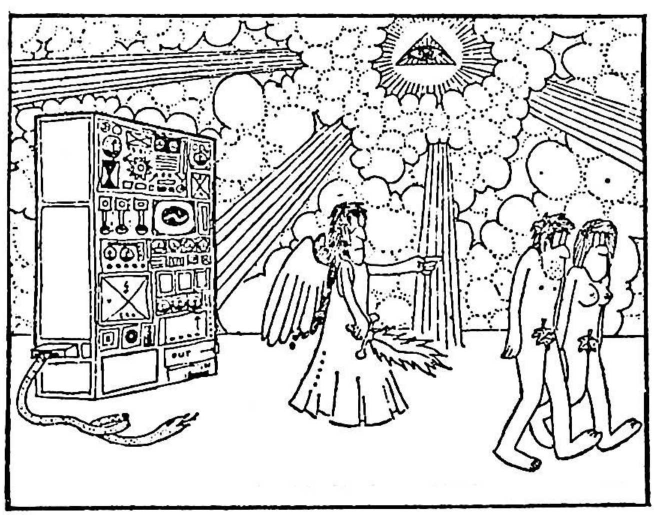
מחיקת צומת שיש לו בן אחד,
למשל 20. תת-העץ השמאלי יראה כך:



מחיקת צומת שיש לו 2 בנים, למשל מחיקה של 42.
האיבר הקטן ביותר בתת-העץ הימני של 42 מועבר
לצומת במקום 42.

תת-העץ הימני יראה כך:





תוכנית 14 (ראה תרגיל תכנות 14 בפרק 16)

```
PROGRAM TEXT_TREE (DATA_FILE, OUTPUT);
CONST
   MAX_CHAR = 15;
TYPE
   WORD = PACKED ARRAY [1..MAX_CHAR] OF CHAR;
   NODE_PTR = ^NODE;
   NODE = RECORD                                   { רשומת איבר בעץ }
             ELEMENT : WORD;                       { האיבר }
             COUNTER : INTEGER;                    { מונה מופעים }
             LEFT_CHILD,                           { מצביע לבן שמאלי }
             RIGHT_CHILD : NODE_PTR                { מצביע לבן ימני }
          END;
   BIN_TREE = NODE_PTR;
VAR
   TREE, TEMP : BIN_TREE;
   DATA_FILE  : TEXT;
   READ_WORD  : WORD;

              { פרוצדורה זו קוראת מלה אחת מהתמליל למערך READ_WORD }

PROCEDURE READ_NEXT_WORD (VAR READ_WORD : WORD);
   VAR
      I : INTEGER;
      CH : CHAR;
   BEGIN
      I := 1;
      READ (DATA_FILE, CH);
      WHILE  NOT (CH  IN ['A'..'Z']) AND NOT EOF (DATA_FILE)  DO
         READ (DATA_FILE, CH);                  { דילוג על תווים שאינם אותיות }
      WHILE  (CH IN ['A'..'Z']) AND NOT EOF (DATA_FILE)  DO
       BEGIN   { קריאת אותיות למערך עד סוף הקובץ, או עד תו שאינו אות }
         READ_WORD [I] := CH;
         I := I + 1;
         READ (DATA_FILE, CH)
       END;
      WHILE  I <= MAX_CHAR  DO
       BEGIN                                    { מילוי רווחים עד סוף המערך }
         READ_WORD [I] := ' ';
         I := I + 1
       END
   END;  { read_next_word }

PROCEDURE MAKE_NULL (VAR TREE : BIN_TREE);                { יצירת עץ ריק }
   BEGIN
      TREE := NIL
   END;    { make_null }
```

```
PROCEDURE INSERT (X    : WORD;                              { הכנסת איבר X לעץ TREE }
               VAR TREE : BIN_TREE);
   BEGIN
      IF  TREE = NIL  THEN
       BEGIN                                                  { המלה אינה מופיעה בעץ }
         NEW (TREE);
         WITH  TREE^  DO
          BEGIN
            ELEMENT := X;
            COUNTER := 1;
            LEFT_CHILD  := NIL;
            RIGHT_CHILD := NIL
          END
       END
      ELSE
         IF  TREE^.ELEMENT > X  THEN          { אם X קטן מהאיבר הנוכחי בעץ }
            INSERT (X, TREE^.LEFT_CHILD)        { הכנס אותו לצד שמאל. }
         ELSE
            IF  TREE^.ELEMENT < X  THEN {אם X גדול מהאיבר הנוכחי בעץ}
              INSERT (X, TREE^.RIGHT_CHILD)    { הכנס אותו לצד ימין. }
            ELSE                                       { האיבר X קיים בעץ. }
               TREE^.COUNTER := TREE^.COUNTER + 1            { הוסף 1 }
                                                     { למונה המופעים. }
   END;  { insert }

PROCEDURE INORDER (TREE : BIN_TREE);                  { הדפסת העץ לפי הסדר }
   BEGIN                                              { ראה הסבר בתוכנית 13 }
      IF  TREE <> NIL  THEN
       BEGIN
         INORDER (TREE^.LEFT_CHILD);
         WRITELN (TREE^.ELEMENT:4, TREE^.COUNTER:8);
         INORDER (TREE^.RIGHT_CHILD)
       END
   END;    { inorder }

BEGIN  { main }
   MAKE_NULL (TREE);                                          { איתחול העץ }
   ASSIGN (DATA_FILE, 'TEXT');                        { פתיחת קובץ התמליל }
   RESET (DATA_FILE);
   WHILE  NOT EOF (DATA_FILE)  DO
    BEGIN
      READ_NEXT_WORD (READ_WORD);                              { קריאת מילה }
      INSERT (READ_WORD, TREE)  {הכנסת המילה לעץ ועדכון מספר ההופעות}
    END;
   CLOSE (DATA_FILE);
   INORDER (TREE)                                              { הדפסת העץ }
END.
```

<u>הערה:</u> פרוצדורות INSERT ו-INORDER מוסברות בתוכנית 13.

תוכנית 15 (ראה תרגיל תכנות 15 בפרק 16)

```
PROGRAM PART_TREE (DATA_FILE, OUTPUT);
CONST
   MAX_PART = 100;
TYPE
   NODE_PTR = ^NODE;
   NAME_TYPE = PACKED ARRAY [1..10] OF CHAR;
   NODE = RECORD                                     { רשומה בעץ }
            NAME,                                    { שם לקוח }
            DATE          : NAME_TYPE;               { תאריך }
            PART          : CHAR;                    { חלק }
            NUMBER        : INTEGER;                 { מספר פעולה }
            LEFT_CHILD,                              { מצביע לבן שמאלי }
            RIGHT_CHILD   : NODE_PTR                 { מצביע לבן ימני }
         END;
   BIN_TREE = NODE_PTR;
   PART_REC = RECORD                                 { רשומת חלק }
                NAME,                                { שם הלקוח }
                DATE    : NAME_TYPE;                 { תאריך }
                PART    : CHAR;                      { חלק }
                NUMBER  : INTEGER                    { מספר חלק }
             END;
VAR
   TREE     : BIN_TREE;
   DATA_FILE : TEXT;
   PARTS    : ARRAY [1..MAX_PART] OF PART_REC;
   LAST     : INTEGER;
   REPORT   : PART_REC;

                    { פרוצדורה לקריאה של נתוני החלקים למערך PARTS,
                      משתנה LAST מחזיר את מספר הרשומות שנקראו. }

PROCEDURE READ_PARTS (VAR LAST : INTEGER);
   BEGIN
      ASSIGN (DATA_FILE, 'TEXT');
      RESET (DATA_FILE);
      LAST := 0;
      WHILE  NOT EOF (DATA_FILE)  DO
       BEGIN
         LAST := LAST + 1;
         WITH  PARTS [LAST]  DO
            READLN (DATA_FILE, NAME, PART, DATE, NUMBER)
       END;
      CLOSE (DATA_FILE)
   END;     { read_parts }

PROCEDURE MAKE_NULL (VAR TREE : BIN_TREE);
   BEGIN
      TREE := NIL
   END;    { make_null }
```

```
                    { פרוצדורה זו בונה את העץ ש-TREE הוא השורש שלו
              { ואיבריו נמצאים במערך בין מקום FIRST לבין מקום LAST.

PROCEDURE BUILD_TREE (VAR TREE  : BIN_TREE;
                          FIRST,
                          LAST  : INTEGER);
   VAR
      MID : INTEGER;
   BEGIN
      IF  FIRST <= LAST  THEN
       BEGIN
         MID := (FIRST + LAST) DIV 2;                    { האיבר האמצעי }
         NEW (TREE);
         WITH  TREE^  DO
          BEGIN                                 { הוספת האיבר האמצעי לשורש }
            NAME := PARTS [MID].NAME;
            PART := PARTS [MID].PART;
            NUMBER := PARTS [MID].NUMBER;
            DATE := PARTS [MID].DATE;
            BUILD_TREE (LEFT_CHILD, FIRST, MID - 1);  { בניית תת-העץ
                                          השמאלי מחצי המערך התחתון. }
            BUILD_TREE (RIGHT_CHILD, MID + 1, LAST)   { בניית תת-העץ
                                           הימני מחצי המערך העליון. }
          END
       END    { if first }
      ELSE
         TREE := NIL                                            { עלה }
   END;    { build_tree }

      {    CODE פרוצדורה זו סורקת את כל העץ לפי סדר מספר הפעולה ולפי
       { ומדפיסה את הרשומות שמתאימות לשדות NAME ו-PART שברשומת REPORT.

PROCEDURE INORDER (TREE   : BIN_TREE;
                   CODE   : CHAR;
                   REPORT : PART_REC);
   BEGIN
      IF  TREE <> NIL  THEN
         WITH  TREE^  DO
          BEGIN
            INORDER (LEFT_CHILD, CODE, REPORT);
            IF ((CODE = 'N') AND (NAME = REPORT.NAME)) OR
               ((CODE = 'P') AND (PART = REPORT.PART)) OR
               (CODE = 'A')  THEN
                  WRITELN (NAME:10, PART:3, DATE:13, NUMBER:5);
                                  {   CODE = 'N'    דו"ח לפי שם:
                                      CODE = 'P'    לפי חלק
                                      CODE = 'A'    כל הפעולות        }
            INORDER (RIGHT_CHILD, CODE, REPORT)
          END    { with }
   END;    { inorder }
```

```
BEGIN  { main }
    MAKE_NULL (TREE);                                      { איתחול העץ }
    READ_PARTS (LAST);                                     { קריאת החלקים למערך }
    BUILD_TREE (TREE, 1, LAST);
    WRITELN ('  לקוח       חלק       תאריך        פעולה');
    WRITELN ('------------------------------------');
    INORDER (TREE, 'A', REPORT);
    WRITELN ('חלק A מופיע בפעולות הבאות:':75);
    REPORT.PART := 'A';
    INORDER (TREE, 'P', REPORT);
    WRITELN ('הפעולות ללקוח בשם רחמים הם:':75);
    REPORT.NAME := 'רחמים     ';
    INORDER (TREE, 'N', REPORT)
END.
```

<u>הסבר:</u>

הפרוצדורה READ_PARTS קוראת את נתוני החלקים למערך PARTS ומחזירה בפרמטר LAST את מיקומו של החלק האחרון במערך (זהו מספר החלקים שנקראו).

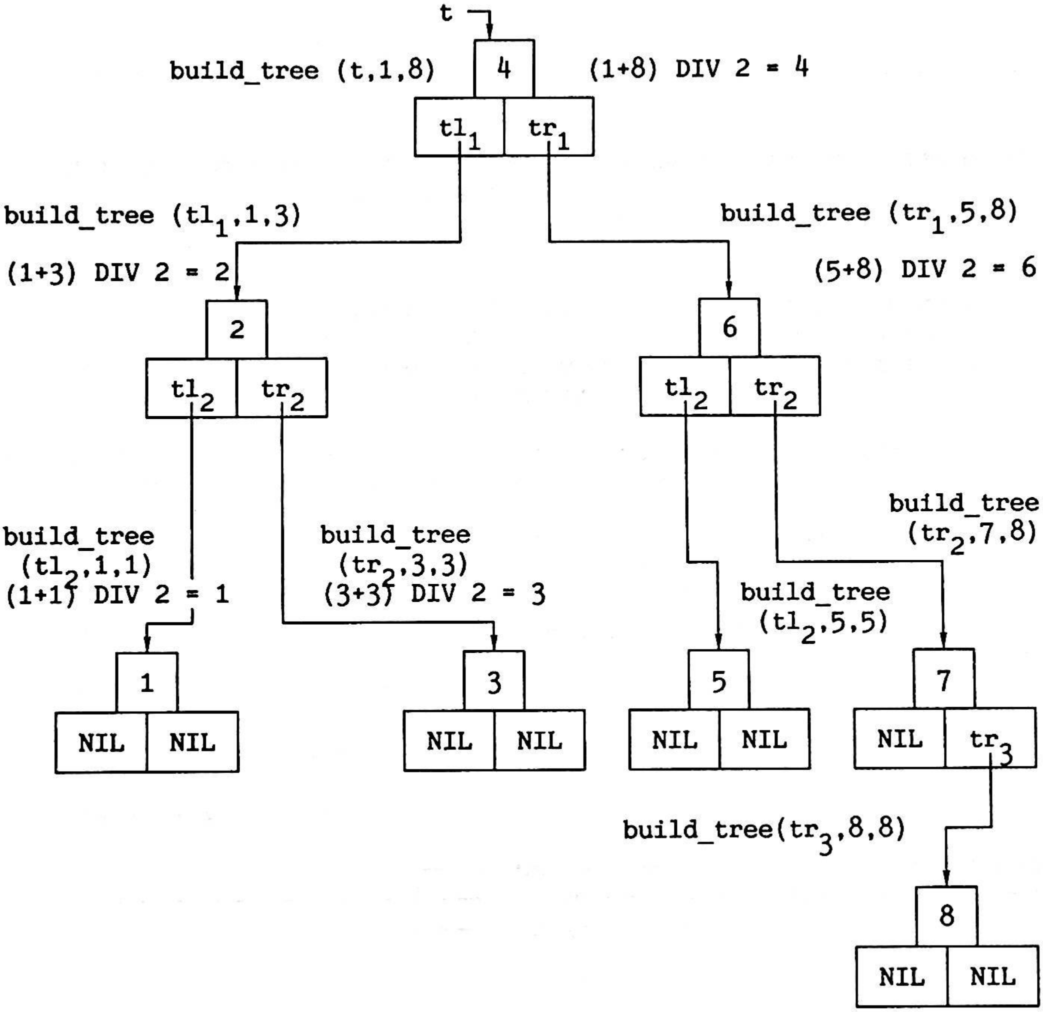
הפרוצדורה BUILD_TREE בונה בצורה רקורסיבית את העץ. היא מקבלת שני פרמטרים המצביעים על גבולות במערך החלקים, שיש לקחת ממנו את האיברים ולבנות מהם עץ. הפרוצדורה מחזירה מצביע לעץ, אשר בשורשו נמצא האיבר האמצעי במערך. תת-העץ הימני מורכב מאיברי המערך הגדולים ממנו ותת-העץ השמאלי מורכב מאיברי המערך הקטנים ממנו.

תת-העץ הימני ותת-העץ השמאלי נבנים ע"י קריאה רקורסיבית לפרוצדורה BUILD_TREE.

תיאור התהליך:

המערך הנתון:

1	2	3	4	5	6	7	8



נספח א' - תרשימי מבנה
(Syntax Diagrams)

בנספח זה מוצגים תרשימי התחביר (Syntax Diagrams) של המשפטים בשפת פסקל הסטנדרטית, כפי שמקובל בשפה האנגלית.

מקרא:

א. יש לקרוא את התרשים משמאל לימין, על פי כיוון החיצים. כאשר החיצים מסתעפים, פירוש הדבר שיש יותר מאפשרות אחת להתקדם. המתכנת יבחר בכיוון הרצוי בהתאם לתנאי הבעיה. יש לשים לב לכך, שניתן לחזור על חלקים של התרשים בהתאם לכיוון החיצים.

ב. כל רישום המופיע במלבן מציין "הגדרה", אשר ניתנת במקום אחר בתוכנית, בפרוצדורה, או בפונקציה.

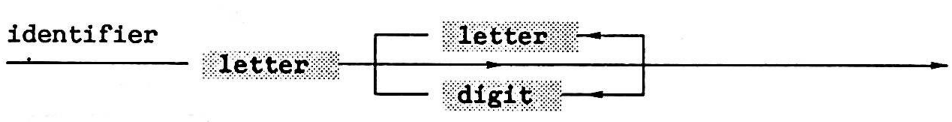
ג. כל רישום שאינו כתוב במלבן, מציין מלה שמורה, או סימן (למשל: . ; : E ועוד), שיש לכתוב אותם כפי שהם מופיעים בתרשים.

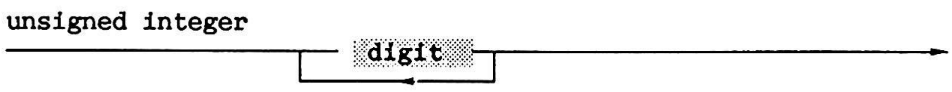
ד. המילים letter, digit המופיעות ללא מלבן מציינות קלט ישיר.

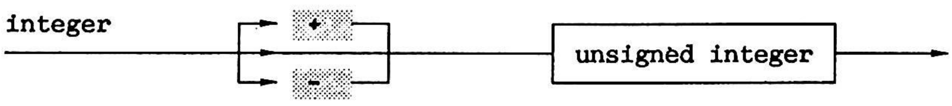
id היא identifier ("מציין").

decl היא declaration.

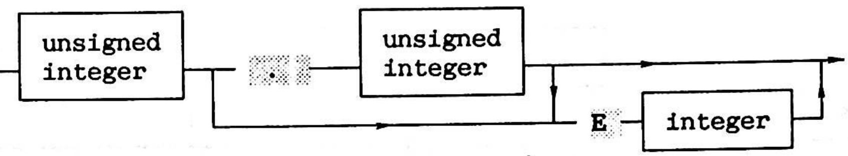
parm היא parameter.



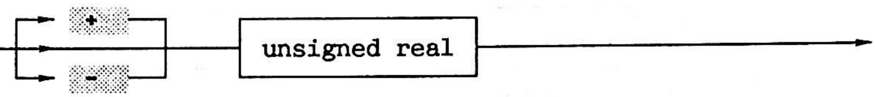




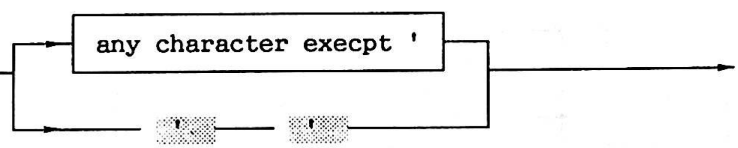
unsigned real



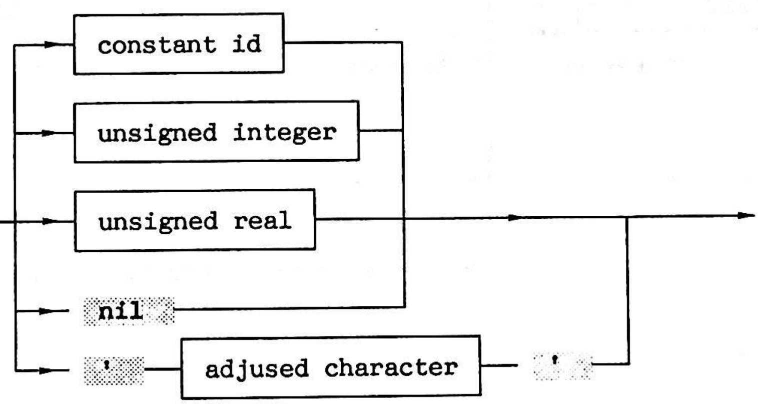
real



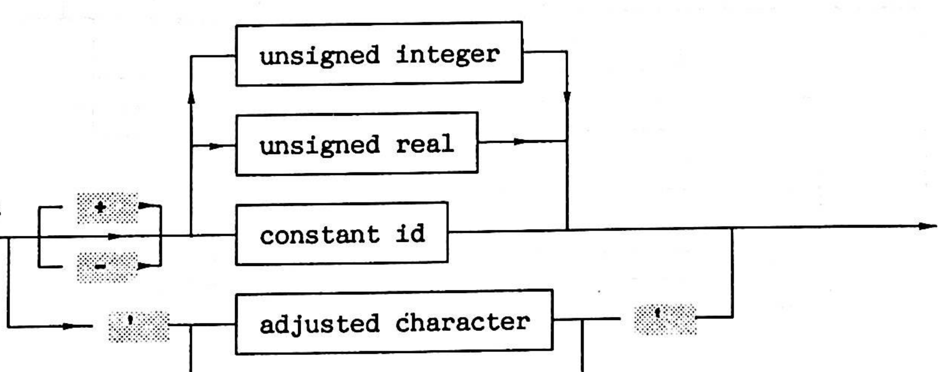
adjusted character

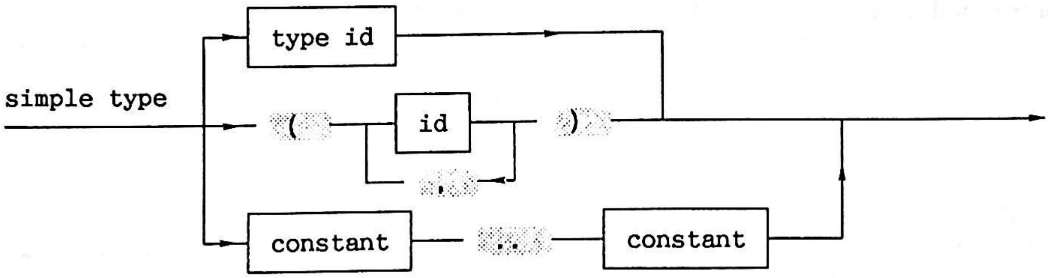


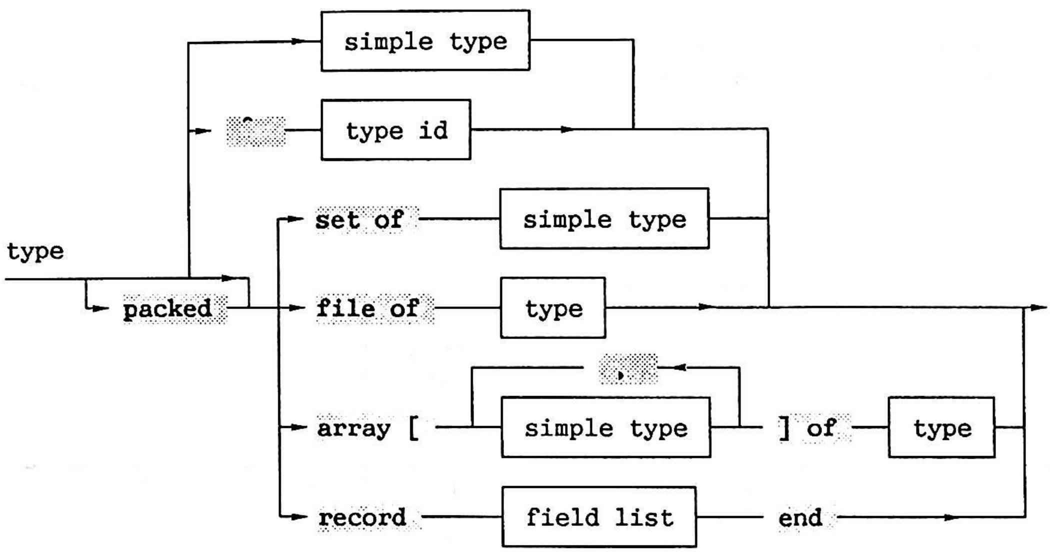
unsigned constant

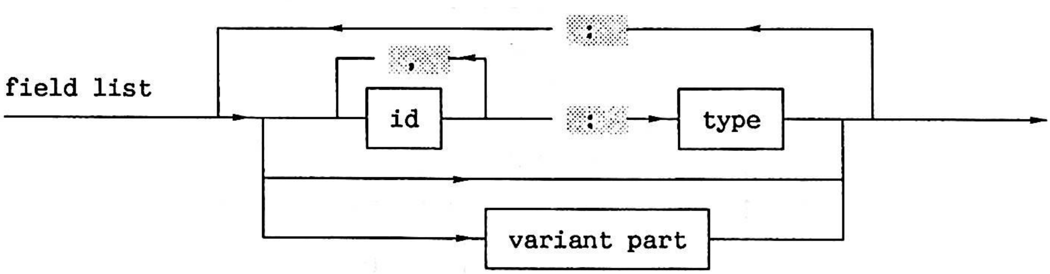


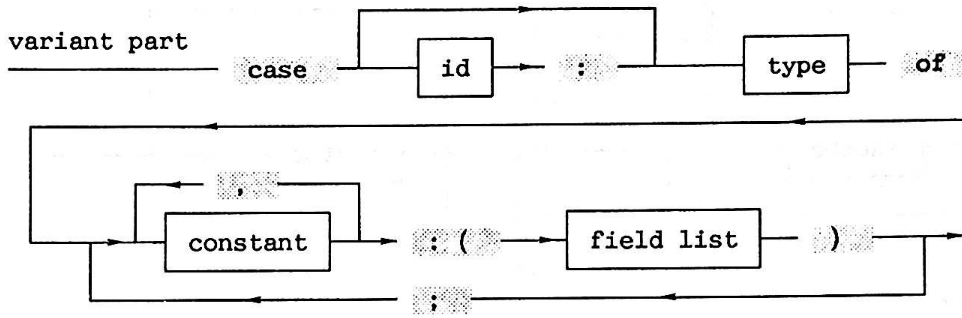
constant

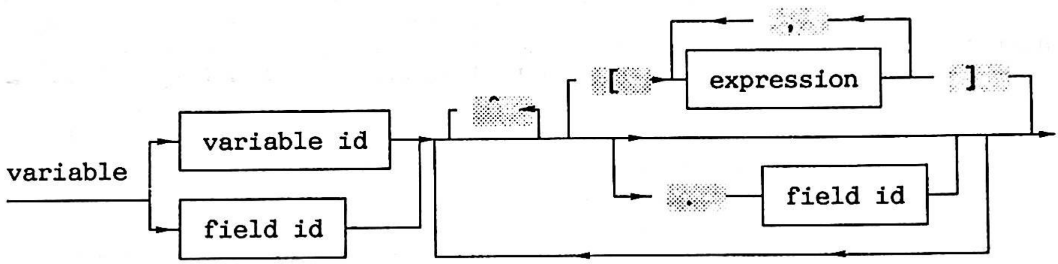


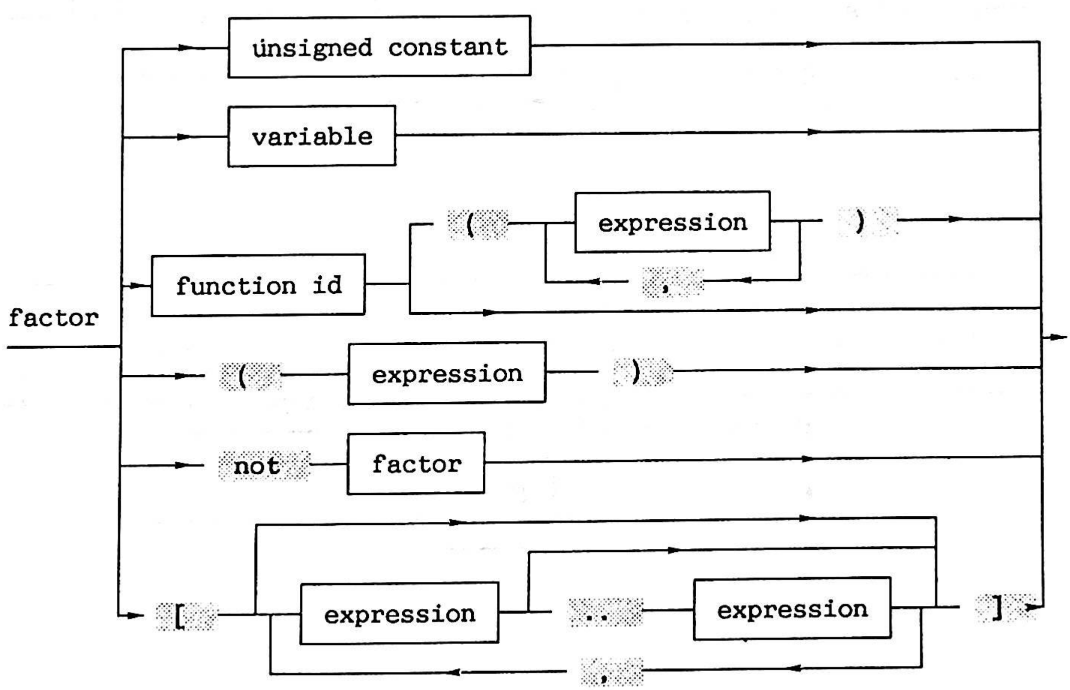

simple type
type id
(
id
)
,
constant
..
constant


type
simple type
^
type id
set of
simple type
packed
file of
type
,
array [
simple type
] of
type
record
field list
end

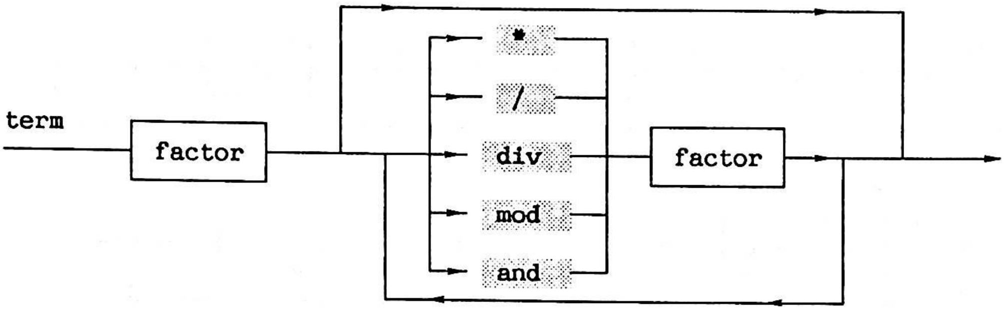

;
,
field list
id
:
type
variant part


variant part
case
id
:
type
of
,
constant
: (
field list
)
;

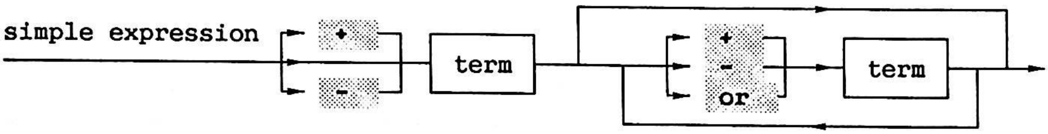

variable
variable id
field id
[
expression
,
]
.
field id


factor
unsigned constant
variable
function id
(
expression
,
)
(
expression
)
not
factor
[
expression
..
expression
,
]

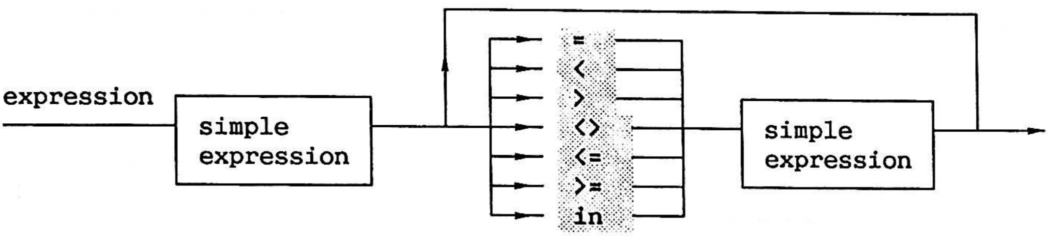
term



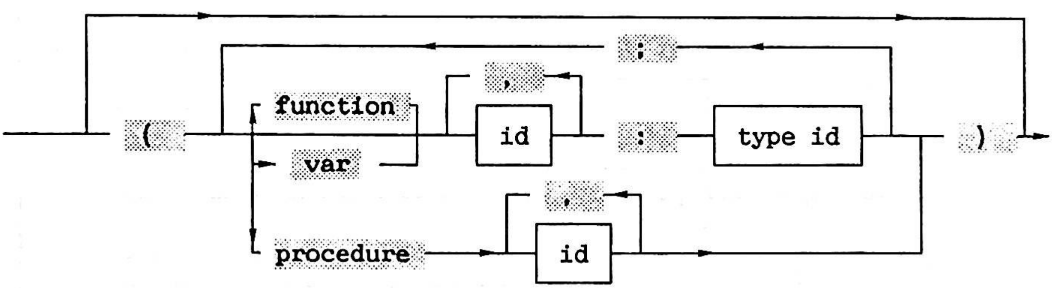
simple expression

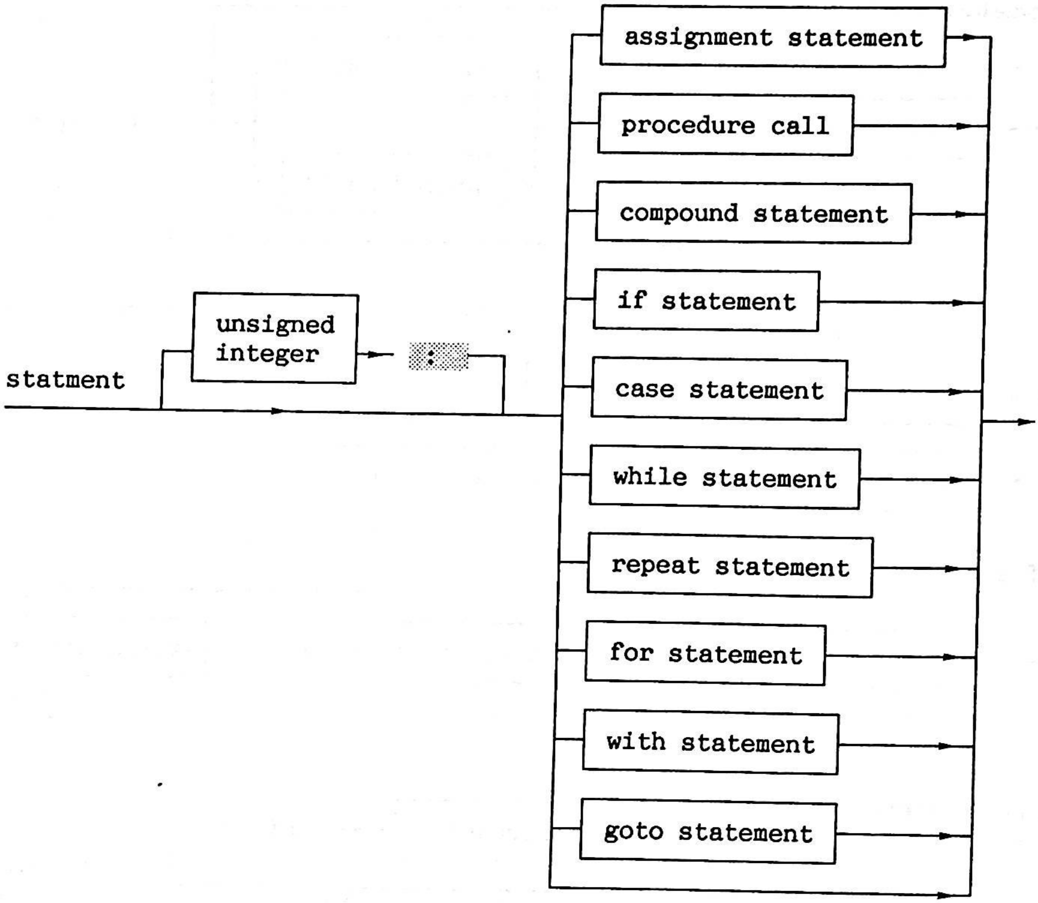


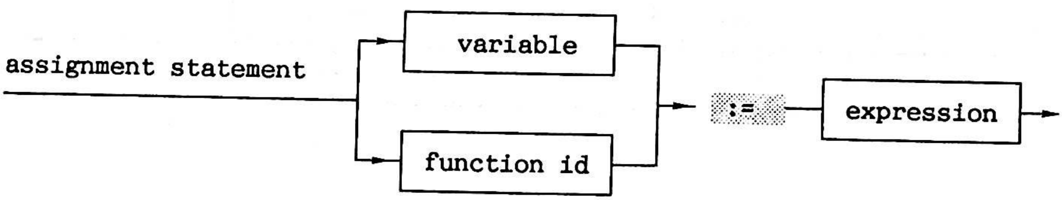
expression



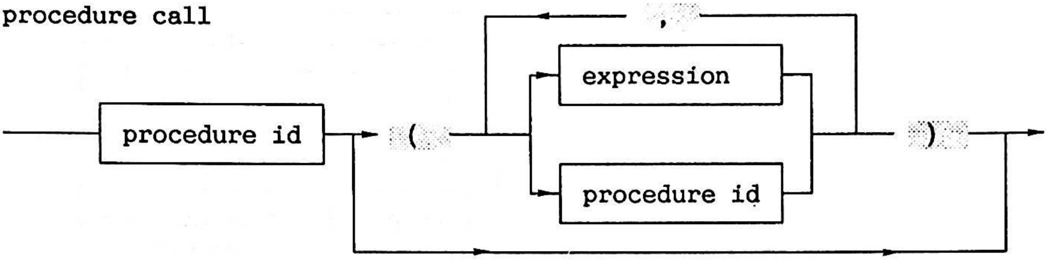
parameter list



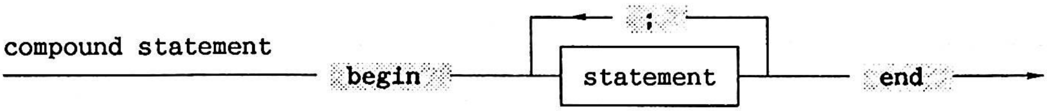

statment
unsigned integer
:
assignment statement
procedure call
compound statement
if statement
case statement
while statement
repeat statement
for statement
with statement
goto statement


assignment statement
variable
function id
:=
expression

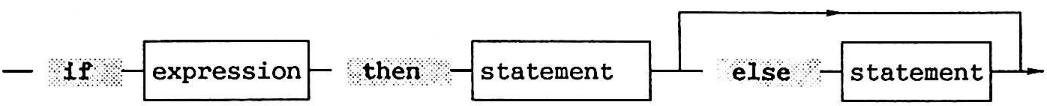
procedure call



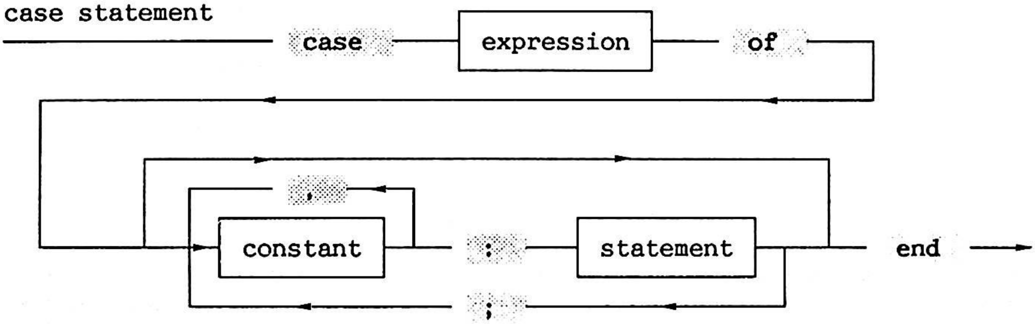
compound statement



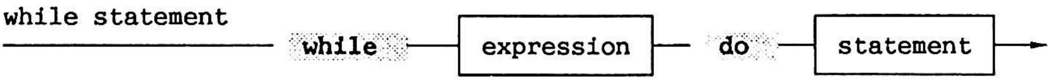
if statement



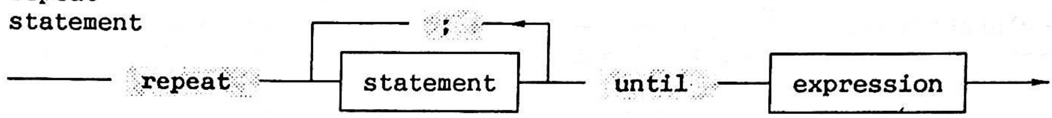
case statement



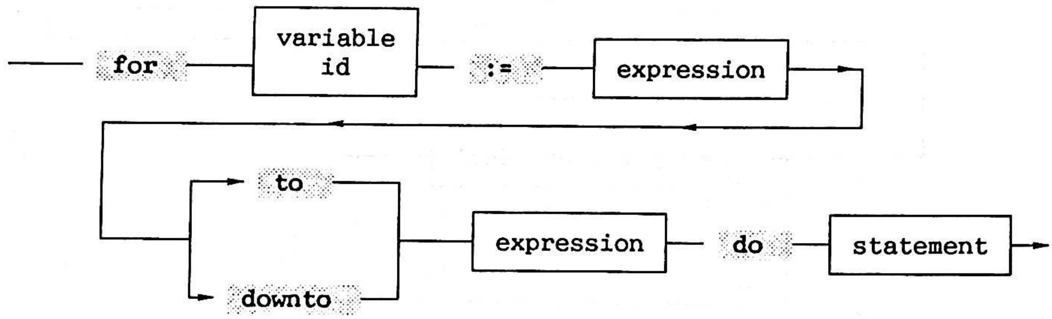
while statement



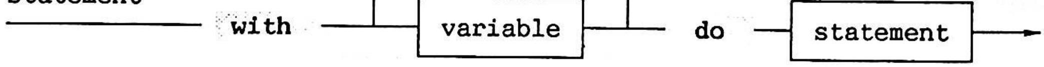
repeat statement



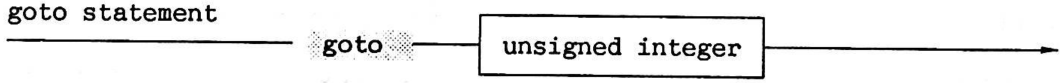
for statement



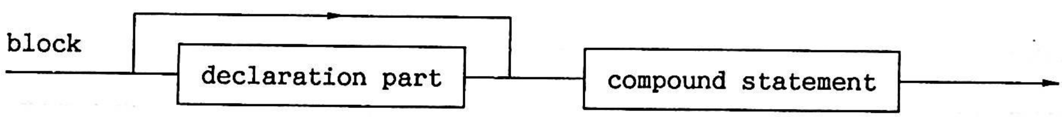
with statement

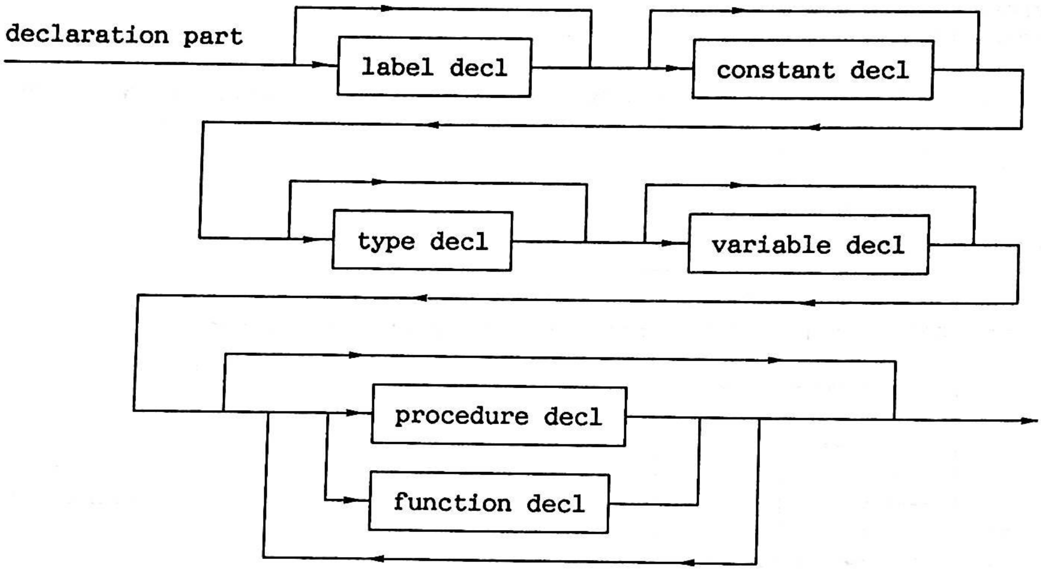


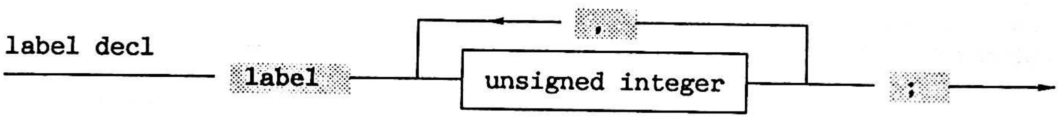
goto statement

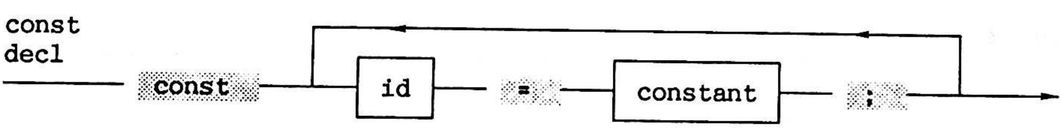


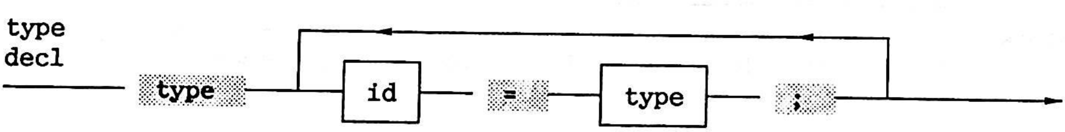
block

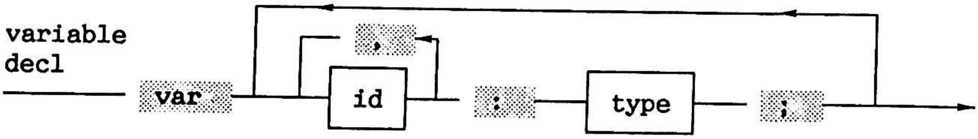



declaration part
label decl
constant decl
type decl
variable decl
procedure decl
function decl

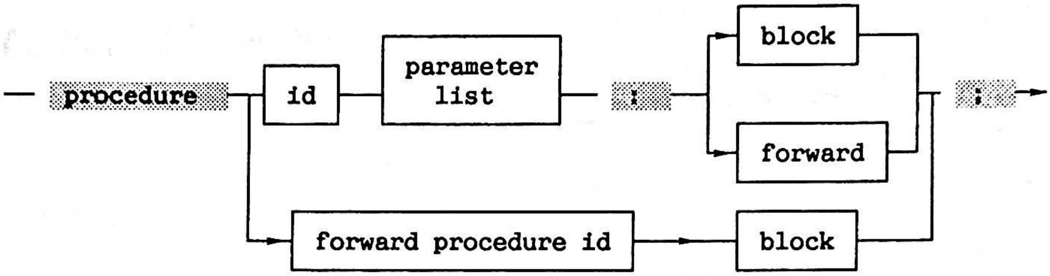

label decl
label
,
unsigned integer
;


const
decl
const
id
=
constant
;

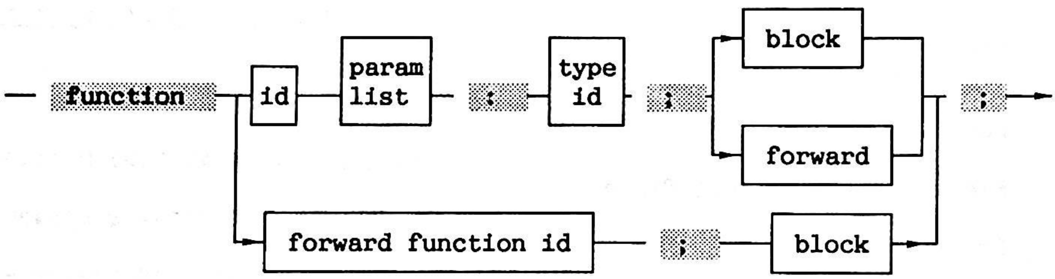

type
decl
type
id
=
type
;


variable
decl
var
,
id
:
type
;

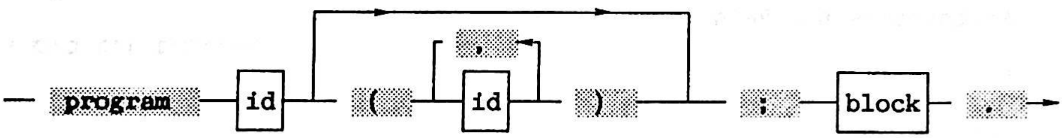
procedure declaration



function declaration



program declaration



record



נספח ב' - שפת פסקל

Quick Reference

<u>תוכן עניינים:</u>

מלים שמורות
Reserved Words

הגדרות בלוק (גוש)
Block Component Identifiers

קבועים מוגדרים מראש
Predefined Constats

איפיונים מוגדרים מראש
Predfined Types

הצהרות לאיפיונים המוגדרים ב-Type
Type Definitions Statements

אופרטורים אריתמטיים ובוליאניים
Arithmetics and Relational Operators

יחסים בין ביטויים
Relational Operators

פעולות בין קבוצות
Set Operators

טבלאות אמת לאופרטורים בוליאניים
Truth Table for: and, or, xor, not

משפטי בקרה
Control Statements

פונקציות אריתמטיות
Arithmetic Functions

פונקציות להסבה ולפעולות סקלרים
Conversion and Scalar Functions

פרוצדורות ופונקציות לטיפול במחרוזות תווים
String-Handling Procedures and Functions

פרוצדורות ופונקציות של קלט/פלט
External Device and File I/O Subprograms

פ/פ להקצאה ולניהול של הזיכרון
Memory Allocation and Manipulation

פרוצדורות ופונקציות לטיפול במסך ובלוח המקשים
Video and Keyboard Subprograms

פקודות לטיפול במסך לפי קוד ANSI
Video Control

פ/פ לשליטה במערכת ההפעלה
System Control Subprograms

פונקציות ופרוצדורות של טורבו-פסקל למחשב IBM-PC
IBM-PC System Subprograms

פ/פ הפועלות ללא כרטיס גרפי
פ/פ הפועלות עם כרטיס גרפי

פקודות ל"צב" במחשב IBM-PC
IBM-PC Turtle Graphics

משתנים בשרות המערכת
System Level Variables

מצביעים לכתובות של פרוצדורות קלט/פלט (טורבו-פסקל בלבד)
I/O Driver Vector Address Variables

מערכים סטנדרטיים
Memory and Port Array Variables

הגדרות מראש לאמצעי קלט/פלט
Text File Predeclared Variables

סימנים מיוחדים·
Special Symbols

הוראות למהדר טורבו-פסקל
Turbo-Pascal Compiler Directives

בנספח זה נשתמש בקיצורים ובסימנים הבאים:

any		כקידומת, שמשמעותה: "כל דבר".
any-type		משתנה מכל איפיון שהוא.
byte	+	משתנה מאיפיון "בית" (byte). בטורבו-פסקל בלבד.
bool		משתנה או ביטוי לוגי/בוליאני (boolean).
chr1		משתנה מסוג "תו" (char).
fil		משתנה מסוג "קובץ" (file).
fp		משתנה מאיפיון "ממשי" (real).
int1		משתנה מסוג "שלם" (integer).
pntr		משתנה מסוג "מצביע" (pointer).
scalar		משתנה מאיפיון "סקלר" (scalar).
scalar-typ		איפיון סקלר, דהיינו איפיון שמוגדר ע"י המתכנת.
str1	+	משתנה מאיפיון "מחרוזת" (string). בטורבו - האיפיון של מחרוזת מוגדרת מראש.
txt		משתנה של קובץ תמליל (text).
memloc		כתובת בזיכרון המחשב.
dos-reg		אוגר של מערכת ההפעלה.
any-string		מחרוזת מכל אורך שהוא.
file-typ		איפיון של קובץ.
[ ]		פרמטר הרשום בתוך הסוגריים הינו אופציונלי (רצוני, לא הכרחי).
intfp		משתנה מאיפיון "ממשי" או איפיון "שלם".
P		פרוצדורה (procedure).
F		פונקציה (function).
פ/פ		קיצור לפרוצדורה או פונקציה.
(רווח)		ה-פ/פ פועלת בפסקל סטנדרטי ובגירסת טורבו-פסקל.
+		ה-פ/פ מתאימה לגירסת טורבו-פסקל בלבד.
-		ה-פ/פ פועלת בפסקל סטנדרטי ואינה בטורבו-פסקל.
*		כאשר המשתנים מהאיפיונים השונים משמשים פרמטר ל-פ/פ, הם יכולים גם להיות ביטויים (פרט למקרה שהפרמטר מועבר כ-Var).

הערה: למהדר טורבו-פסקל ישנן מספר גירסאות. נספח זה מביא את הפונקציות המתאימות לגירסה 3.0.

מלים שמורות
Reserved Words

absolute +	לקביעת כתובת של משתנה בזיכרון.
and	אופרטור בוליאני ואריתמטי. בטורבו - מתאים גם למשתנים מאיפיון שלם או בית.
array	הצהרה על מערך (וקטור).
begin	סימון תחילתו של גוש פקודות (בלוק) אשר תוחמים אותו במלה end. מעתה נסמן גוש של פקודות במונח: פקודה מורכבת.
case	פקודה לשליטה במבנה הזרימה של פקודות בהתאם למספר אפשרויות, אשר תוחמים אותן במלה end. היא משמשת גם להצהרה על רשומה בעלת אורך משתנה. מילים נוספות בשימוש: else, of.
const	הצהרה על קבועים בתחילת תכנית או פ/פ.
div	פעולת חילוק במספרים שלמים. גם התוצאה היא מספר שלם.
do	לשימוש בסוף השורה של הצהרה על לולאת while, ובשילוב עם for ו-with.
downto	בהצהרה של לולאת for: מונה הלולאה פוחת כל פעם ב-1.
else	בפקודת תנאי (if): המלה מקדימה את הפקודה המורכבת שמתבצעת כאשר התנאי איננו מתקיים. בשילוב גם עם case.
end	סימון סיום של פקודה מורכבת (begin), של פקודת case ושל הצהרת record.
external +	הצהרה על הכללה של תת-תכנית חיצונית בשפת מכונה, בגוף תכנית הפסקל.
file	הצהרה על איפיון של קובץ.
forward	הצהרה שתוכן פ/פ יופיע בשלב מאוחר יותר של התכנית. כותרת ה-פ/פ מופיעה בסמוך להצהרה זו.
for	פקודת בקרה: לולאה מבוקרת בהתאם למונה הלולאה (to, do, downto).
function	הצהרה על פונקציה.
goto	סיעוף ממקום אחד בתכנית למקום אחר בתכנית (בו ישנה תוית).
inline +	שילוב של שגרות בשפת מכונה בתכנית פסקל.

if	פקודת תנאי לשליטה על מבנה הזרימה של הפקודות. בשימוש עם המלים then ו-else.
in	אופרטור לקבוצות, למשל "האם פרט/ים מוכלים בקבוצה?".
label	הצהרה בתחילת תכנית (או פ/פ) על שלמים (בטורבו גם מחרוזות) שישמשו כתויות בתכנית עבור פקודת ה-goto.
mod	אופרטור בין מספרים שלמים, לקבלת שארית החילוק.
nil	קבוע שערכו שווה לקבוצה ריקה, או מצביע שאין לו ערך.
not	אופרטור בוליאני. בטורבו-פסקל גם אופרטור על ביטויים מאיפיון שלם או בית.
overlay +	בטורבו-פסקל - להצהרה על כך שכמה פ/פ ישתמשו באותו שטח זיכרון לחילופין, למקרה שאי אפשר לכלול את כולן בזיכרון מפאת חוסר מקום.
of	לשימוש בפקודת case, בהצהרה על מערך, קובץ וקבוצה.
or	אופרטור בוליאני. בטורבו-פסקל - גם אופרטור אריתמטי בין ביטויים שאיפיונם שלם או בית.
packed	הצהרה על מערכים דחוסים. זהו גם שם של פרוצדורה שדוחסת מערך. בטרובו-פסקל מתבצעת דחיסה אוטומטית בכל מקום שניתן, ולכן השימוש ב-packed מיותר (אך אין זו שגיאה).
procedure	הצהרה על פרוצדורה (תת-תכנית - שגרה).
program	הצהרה על תחילת התכנית. הצהרה זו אינה הכרחית בטורבו-פסקל.
record	הצהרה על איפיון רשומות. בסיום הצהרה יש לרשום end.
repeat	שליטה לוגית. לולאה שמתבצעת עד אשר תנאי מסוים מתקיים (until).
set	הצהרה על איפיון של קבוצה.
shl +	אופרטור על ביטויים מאיפיון בית (byte) או שלם (integer).
shr +	אופרטור על ביטויים או משתנים מאיפיון בית או שלם.
string +	הצהרה על איפיון של מחרוזת דינמית.
then	שימוש בפקודת התנאי if (בסיום הגדרת תנאי) ומעבר למצב TRUE.
type	הצהרה בתחילת תכנית או פ/פ, להגדרת איפיונים חדשים.
to	לשימוש בלולאת for, כאשר המונה גדל ב-1 בכל מעבר בלולאה.

until   לסימון סיומה של לולאת ה-repeat.

var   הצהרה על משתנים בתחילת תכנית או פ/פ.

while   ממשפטי הבקרה בתכנית. לולאה שמתבצעת כל עוד תנאי מסוים מתקיים.

with   הצהרה על פנייה לשדות ברשומה. בשימוש עם do.

xor +   אופרטור בוליאני. בטורבו גם אריתמטי בין משתנים שאיפיונם בית או שלם.



הגדרות בלוק (גוש)
Block Component Identifiers

program	הצהרה על תחילת התכנית. הצהרה זו אינה הכרחית בטורבו-פסקל.
label	הצהרה על שימוש בקבועים שלמים כתויות בתכנית או ב-פ/פ. בטורבו-פסקל התויות יכולות להיות גם מחרוזות.
const	הצהרה על קבועים בתכנית או ב-פ/פ.
type	הצהרה על איפיונים חדשים בתחילת תכנית או פ/פ.
var	הצהרה על משתנים בתחילת תכנית או פ/פ.
procedure	הצהרה על תחילת פרוצדורה (תת-תכנית).
function	הצהרה על תחילת פונקציה (תת-תכנית שמחזירה ערך).
begin-end	הגדרה של גוש פקודות (מלים אלו תוחמות את תחילת גוש.הפקודות וסיומו).

הערה:
סדר ההצהרות של חלקי התוכנית (בלוקים או גושים) הינו סדר הכתיבה המחייב בפסקל סטנדרטי. בטורבו-פסקל אין חובה לשמור על סדר זה, אך יש לשים לב לא להשתמש במזהה (Identifier) לפני שהוגדר.

קבועים מוגדרים מראש
Predefined Constats

MAXINT	השלם הגדול ביותר שקיים במחשב. בטורבו-פסקל ערך הקבוע הוא 32767 (16 סיביות).
NIL	קבוע שמסמן קבוצה ריקה, או מצביע ללא ערך.
PI +	קבוע ממשי שערכו שווה ל-π, דהיינו 3.1415926536.
TRUE & FALSE	קבועים שמשמשים כערכים לביטויים בוליאניים.
מספרים	קיימים קבועים שלמים (בטורבו - גם איפיון "בית"), וקבועים ממשיים בייצוג של בסיס עשרוני ובסיס 16.
תוים	קיימים קבועים שהם תוים בודדים (שאיפיונם char) וקבועים שהם מחרוזות (שאיפיונם מערך של תוים). ניתן לייצג תוים שאינם סטנדרטיים, כלומר - אינם קיימים על לוח המקשים.

איפיונים מוגדרים מראש
Predfined Types

Boolean	איפיון בוליאני. משתנה מאיפיון זה יכול לקבל את הערכים True או False בלבד. ניתן גם להציב בו ביטויים בוליאניים כמו בפקודת ה-if.
+ Byte	איפיון של מספר שלם ללא סימן, בתחום 0 עד 255 (בית אחד בן 8 סיביות).
Char	איפיון של תווים, אשר מיוצגים בד"כ בשיטת ASCII. קיימים עד 256 תווים שונים.
Integer	ערכים שלמים. בטורבו פסקל טווח הערכים הוא: ‎-32768 עד ‎+32767 (2 בתים, שהם 16 סיביות).
Real	מספרים ממשיים.
+ String[n]	איפיון למחרוזת דינמית, שאורכה משתנה. n הוא מספר התווים המכסימלי במחרוזת.
Text	איפיון לקובץ תמליל חיצוני שמכיל תווים בלבד.
+ Text[nn]	איפיון לקובץ תמליל, שבו מגדירים את גודל המאגר (Buffer) שיהיה לקובץ. גודלו יהיה תת כפולות של 128 בתים.

הצהרות לאיפיונים המוגדרים ב-Type
Type Definitions Statements

array of	איפיון של מערך.
case of	הגדרה של קטע של רשומה בעלת אורך משתנה.
file of	איפיון של קובץ.
packed	הצהרה על מערכים דחוסים. זהו גם שם של פרוצדורה שדוחסת מערך. בטרובו-פסקל מתבצעת דחיסה אוטומטית בכל מקום שניתן, ולכן השימוש ב-packed מיותר (אך אין זו שגיאה).
record-end	איפיון של רשומה. מלים אלו תוחמות את תחילת איפיון השדות ברשומה ואת סופם.
set of	איפיון של קבוצה.

אופרטורים אריתמטיים ובוליאניים
Arithmetics and Relational Operators

-	מינוס אונארי (שפועל על נפעל יחיד). משנה את סימנו של ביטוי או משתנה בעל איפיון שלם או ממשי.
not	אופרטור בוליאני. הופך את ערך הנפעל מ-true ל-false ולהיפך. + אופרטור על משתנים שלמים ומסוג בית. גורם להפיכת סיבית 1 ל-0 ולהיפך.
*	פעולת כפל על איפיונים מסוג ממשי ושלם.
/	פעולת חילוק שהתוצאה שלה היא מספר ממשי.
div	פעולת חילוק בין שלמים. תוצאת הפעולה הינה תמיד מספר שלם.
mod	אופרטור בין שלמים. התוצאה היא שלם שהוא שארית החלוקה ביניהם.
and	אופרטור בוליאני. אם ערך שני הביטויים הוא true - תוצאת הפעולה היא true; אחרת - תוצאת הפעולה היא false. + אופרטור בין משתנים מאיפיון בית או שלם. פעולת ה-and בודקת אם יש התאמה בין כל שתי סיביות בשני המשתנים. רק אם בשניהם תהיינה סיביות 1 במקומות המקבילים - התוצאה תהיה 1 במקום המתאים במשתנה התוצאה.
shl +	הזזה שמאלה של הסיביות במשתנה מאיפיון בית או שלם. המשמעות - כפל בחזקות של 2.
shr +	הזזה ימינה של הסיביות במשתנה מאיפיון בית או שלם. המשמעות - חלוקה בחזקות של 2.
+	פעולת חיבור. אופרטור זה פועל בין מספרים שלמים, ממשיים ומסוג בית.
-	פעולת חיסור. אופרטור זה פועל בין מספרים שלמים, ממשיים ומסוג בית.
or	אופרטור בוליאני. אם הערך של אחד מהביטויים הינו true - אזי תוצאת הפעולה היא true. + אופרטור בין ביטויים מאיפיון בית או שלם. הפעולה מתבצעת בהתאמה בין כל שתי סיביות בשני האופרנדים; אם אחד מהם שווה ל-1, הסיבית המתאימה בתוצאה תהיה גם כן 1; אחרת היא תהיה 0.
xor +	אופרטור בוליאני. אם הערך של שני האופרנדים שונה, האחד מהם true - התוצאה תהיה true (1), ואם ערכם שווה - התוצאה תהיה false (0).

יחסים בין ביטויים
Relational Operators

val1 = val2 — יחס של שיוויון: האם יש שיוויון?

val1 <> val2 — האם שני הביטויים שונים זה מזה?

val1 < val2 — האם הערך של val1 קטן מהערך של val2?

val1 <= val2 — האם הערך של val1 קטן או שווה לערך של val2?

val1 > val2 — האם הערך של val1 גדול מהערך של val2?

val1 >= val2 — האם הערך של val1 גדול או שווה לערך של val2?

הערה:

val1 ו-val2 חייבים להיות מאותו איפיון.

אסור שהאיפיון יהיה מסוג רשומה או וקטור (יוצא מהכלל הוא וקטור של תווים: array of char).

התוצאה של יחס בין ביטויים בוליאניים הינה True, כאשר התנאי מתקיים. התוצאה תהיה False, כאשר היחס אינו מתקיים.

פעולות בין קבוצות
Set Operators

set1 * set2 — חיתוך בין קבוצות.

set1 + set2 — איחוד בין קבוצות.

set1 - set2 — חיסור בין קבוצות.

set1 = set2 — יחס: true - אם קיים שיויון בין קבוצות.

set1 <> set2 — יחס: true - אם קיים שוני בין קבוצות.

set1 <= set2 — יחס: true - אם הקבוצה הראשונה מוכלת בשניה.

set1 >= set2 — יחס: true - אם הקבוצה השניה מוכלת בראשונה.

set1 in set2 — יחס: אם הקבוצה הראשונה מוכלת בקבוצה השניה (מצב =>), אך במקרה זה set1 אינו חייב להיות קבוצה, ויכול להיות גם איבר בודד.

הערה: set1 ו-set2 צריכות להיות קבוצות מאותו איפיון.

טבלאות אמת לאופרטורים בוליאניים
Truth Table for: and, or, xor, not

טבלת אמת עבור האופרטור And

Val1	Val2	Result
false	false	false
false	true	false
true	false	false
true	true	true

טבלת אמת עבור האופרטור Or

Val1	Val2	Result
false	false	false
false	true	true
true	false	true
true	true	true

טבלת אמת עבור האופרטור Xor

Val1	Val2	Result
false	false	false
false	true	true
true	false	true
true	true	false

טבלת אמת עבור האופרטור Not

Val	Result
false	true
true	false

<u>הערה:</u> אופרטור זה פועל על משתנה אחד בלבד.

משפטי בקרה
Control Statements

begin-end	תיחום של גוש פקודות, או פקודה מורכבת.
case of-else	בהתאם לערך של המשתנה ב-case יתבצע אחד מגושי הפקודות שמוגדרים (סיעוף רב-נתיבי). ה-else הוא אופציונלי ומופיע אחרון מבין האפשרויות של ה-case, לפני הפקודה המורכבת שתבוצע אם משתנה ה-case לא היה שווה לאף אחד מן הערכים שציינו. ה-else לא קיים בכל הניבים של הפסקל. לעתים מופיעה במקומו המלה Otherwise.
exit +	חזרה אל התכנית הקוראת מתוך לולאות while, repeat, ו-for.
for-do	לולאה מבוקרת בראשה בעזרת מונה. ניתן להוסיף בכל פעם למונה הלולאה 1 או 1-.
goto	קפיצה למקום בתכנית בו נמצאת התוית שבפקודת ה-goto.
if-then-else	ביצוע פקודה מורכבת במקרה שהתנאי של ה-if מתקיים. ע"י שימוש ב-else אפשר לבצע פקודה או גוש פקודות נוסף במקרה שהתנאי איננו מתקיים (סיעוף דו-נתיבי).
repeat-until	לולאה שמתבצעת עד אשר תנאי מסוים מתקיים. גוף הלולאה יתבצע לפחות פעם אחת, כי הבקרה נמצאת בסופה.
while-do	לולאה שמבוקרת בראשה ומתבצעת כל עוד תנאי מסוים מתקיים. אם התנאי איננו מתקיים כבר בתחילה, גוף הלולאה לא יבוצע אף לא פעם אחת.
with-do	הצהרה לפקודה מורכבת שבה מופיע שם רשומה. היא מאפשרת לפנות לשדות של רשומה מבלי לציין את שמה.

פונקציות אריתמטיות
Arithmetic Functions

הפרמטר intfp מכוון למספר שלם או ממשי.
הפרמטר fp מכוון למספר ממשי בלבד.

Abs(intfp)	הפ' מחזירה ערך מוחלט של הפרמטר.
ArcTan(intfp)	הזוית מחושבת לפי הפרמטר, שהוא הטנגנס שלה.
+ Cos(intfp)	קוסינוס הזוית.
Exp(intfp)	העלאה של e בחזקה של הפרמטר.
+ Frac(intfp)	חלק השבר של המספר (החלק שמימין לנקודה העשרונית) המיוצג ע"י הפרמטר.
Int(intfp)	החלק השלם של המספר המיוצג ע"י הפרמטר. הפונקציה מחזירה ערך ממשי (זהו ההבדל מהפונקציה Trunc).
Ln(intfp)	לוגריתם טבעי של הפרמטר.
Random	מספר אקראי, אשר שווה או גדול מ-0 וקטן מ-1.
Random(int1)	מספר אקראי, אשר שווה או גדול מ-0 וקטן מהפרמטר של הפונקציה (מספר שלם בלבד).
Round(fp)	מחזיר מספר שלם, שהוא עיגול של מספר ממשי.
Sin(intfp)	סינוס הזוית.
Sqr(intfp)	העלאה בריבוע של הפרמטר.
Sqrt(intfp)	שורש ריבועי של המספר.
Trunc(fp)	קיצוץ חלק השבר של המספר הממשי. הפונקציה מחזירה ערך שלם.

<u>הערה:</u> כל הפונקציות הטריגונומטריות מחושבות לגבי זויות ברדיאנים.
1 מעלה = 0.17453 רדיאן.
1 רדיאן = 57.29577 מעלות.

בפונקציות Exp ו-Ln הערך של e הוא: e=2.718282 (e - בסיס של לוגריתם טבעי).

פונקציות להסבה ולפעולות סקלרים
Conversion and Scalar Functions

Chr(int1)	הפ' מחזירה תו לפי קוד ה-ASCII שלו, על פי הערך של הפרמטר.
Odd(int1)	פונקציה בוליאנית, אשר מחזירה true כאשר המספר הוא אי-זוגי.
Ord(scalar)	הפ' מחזירה את מספרו הסידורי של הסקלר מאותו איפיון.
+ Ord(pntr)	במערכת CP/M-80 בלבד: הפ' מחזירה ערך שלם שהוא הכתובת שמכיל המצביע.
Pred(scalar)	הפ' מחזירה ערך הקודם לפרמטר בהתאם לאיפיונו.
Succ(scalar)	הפ' מחזירה ערך העוקב לפרמטר בהתאם לאיפיונו.
+ Scalar—typ (int1)	הפ' מחזירה ערך באיפיון של שם הפונקציה. הערך המוחזר יהיה הערך שמספרו הסידורי באיפיון הוא int1. מתקיים: Ord(Scalar_typ(int1))=int1
+ Scalar_typ (scalar)	הפ' מחזירה ערך באיפיון של שם הפונקציה. הערך המוחזר יהיה הערך שמספרו הסידורי באיפיון Scalar_typ, זהה למספרו הסידורי של scalar באיפיון שלו (האיפיון של Scalar_typ שונה מהאיפיון של scalar). מתקיים: Scalar_typ(scalar)=Scalar_typ(Ord(scalar))

פרוצדורות ופונקציות לטיפול במחרוזות תווים
String-Handling Procedures and Functions

F + Concat (str1,str2[,str3...])	צירוף של מחרוזות. ניתן לחבר מחרוזות גם בעזרת האופרטור +.
F + Copy(str1,int1,int2)	העתקת של int2 תווים מהמחרוזת str1, החל ממקום int1.
F + Insert(str1,str2,int1)	החדרה של מחרוזת str1 לתוך מחרוזת str2, החל ממקום int1.
F + Length(str1)	הפ' מחזירה את אורכה של המחרוזת.
F + Pos(str1,str2)	מיקומה של תת-מחרוזת str1 בתוך מחרוזת str2.
F + UpCase(chr1)	הפיכת תו מאות קטנה לאות גדולה (Capital Letter).
P + Str(int1,str1)	הסבה של מספר (שלם או ממשי) למחרוזת תווים.
P + Val(str1,int1,int2)	הסבה של מחרוזת str1 למספר int1. int2 הינו דגל שמציין את הצלחתה או אי הצלחתה של פעולה זו.

<u>הערה:</u> בפסקל סטנדרטי, "מחרוזת" היא מערך דחוס של תווים. השפה אינה כוללת פ/פ לטיפול במחרוזות.
הפעולות בין המחרוזות, אשר אפשריות בפסקל סטנדרטי הן:
א. הצבת ערך למחרוזת.
ב. השוואה בין מחרוזות (לפי הערך של התווים).
ג. קלט ופלט של מחרוזות.

פרוצדורות ופונקציות של קלט/פלט
External Device and File I/O Subprograms

P +	Append(txt)	פתיחת קובץ תמליל להוספה.
P +	Assign(fil,str1)	ייחוס שם קובץ שעל הדיסק למשתנה בתכנית בעל איפיון של קובץ.
P +	BlockRead (fil,any,int1[,int2..])	קריאה מקובץ חסר-איפיון אל משתנה. פרוצדורה זו נועדה לקלט מהיר בדרך פשוטה.
P +	BlockWrite (fil,any,int1[,int2..])	כתיבת ממשתנה לקובץ חסר-איפיון. מיועדת לפלט מהיר בדרך פשוטה.
P +	Chain(chn1)	טעינה לזיכרון והרצת קובץ CHN.* של הטורבו-פסקל, שעבר הידור (קומפילציה). כאשר התוכנית גדולה מ-64K (מגבלת הטורבו) ניתן לפצל אותה למספר תוכניות ולהעביר את הביצוע מתוכנית לתוכנית.
P	Close(fil)	סגירת קובץ חיצוני. טוב גם עבור קובץ תמליל.
F	Eof(fil)	מחזירה ערך TRUE במצב של סוף קובץ, ו-FLASE - אם לא. הפונקציה פועלת גם על קובץ תמליל.
F	Eoln(txt)	מחזירה ערך TRUE בקריאה של סוף שורה בקובץ תמליל, אחרת - FALSE.
P +	Erase(fil)	מחיקת קובץ מקטלוג מערכת ההפעלה.
P +	Execute(fil)	טעינה והרצה של תכנית שעברה הידור.
F +	Filepos(fil)	מחזירה את מיקומו של מצביע הקובץ (ביחס לתחילתו).
P +	Flush(fil)	כתיבת המאגר של הקובץ לדיסק.
P -	Get(fil[,any])	העברת רשומה מהקובץ לזיכרון. פרוצדורה זו אינה פועלת בטורבו-פסקל.
F +	LongFilePos(fil)	כמו הפונק' FilePos, עבור קובץ שגדול מ-32Kb.
F +	LongFileSize(fil)	כמו הפונק' FileSize, עבור קובץ שגדול מ-32Kb.
P +	LongSeek(fil,fp)	הצבת מצביע הקובץ בקובץ שגדול מ-32Kb.

Put(fil[,any])	-	P	העברה מהזיכרון לקובץ. פרוצדורה זו אינה פועלת בטורבו-פסקל.
Read ([fil,]any1[,any2..])		P	קלט מקובץ שעל הדיסק. הפרוצדורה Read פועלת בפסקל סטנדרטי על קובץ תמליל בלבד, ובטורבו - על כל קובץ שהוא.
Readln		P	מעבר לשורת הקלט הבאה.
Readln ([txt,]str1[,str2..])		P	קלט שורה של קובץ תמליל ודילוג להתחלת שורת הקלט הבאה.
Rename(fil,str)	+	P	שינוי שם של קובץ שעל הדיסק.
Reset(fil)		P	פתיחת קובץ לקריאה. הפונ' פועלת גם על קבצי תמליל.
Rewrite(fil)		P	יצירה ופתיחה של קובץ חדש על הדיסק לכתיבה. מיועדת גם לקבצי תמליל.
Seek(fil,n)	+	P	הצבת מצביע הקובץ במקום n ביחס לתחילת הקובץ.
Seekeoln(txt)	+	P	הצבה של מצביע הקובץ בסוף השורה.
Seekeof(txt)	+	P	הצבה של מצביע הקובץ בסוף הקובץ.
Truncate(fil)	+	P	השמטה של חלק הקובץ, מהמקום בו נמצא המצביע ועד סוף הקובץ.
Write ([fil,]any1[,any2..])		P	כתיבה בקבצים. הפרוצדורה Write פועלת בפסקל סטנדרטי על קובץ תמליל בלבד. ובטורבו - כל כל קובץ שהוא.
Writeln		P	מעבר לשורת הפלט הבאה.
Writeln ([txt,]any1[,any2..])		P	פלט לשורה ומעבר לשורה חדשה בקבצי תמליל (רק למשתנים בעלי איפיון סטנדרטי (ראה בפקודת Write).

<u>הערה:</u> אם הפרמטר של פ/פ הוא fil, וה-פ/פ פועלת גם על קבצי תמליל - הדבר הוזכר במפורש. אחרת - ה-פ/פ פועלת רק על קובץ רגיל.

פ/פ להקצאה ולניהול של הזיכרון
Memory Allocation and Manipulation

Addr(any)	F +	את הכתובת בזיכרון של כל משתנה או של מצביע על אופרנד הפונקציה.
Dispose(pntr)	P	שיחרור מקום בזיכרון שהוקצה ל-pntr ע"י הפרוצדורה New.
FreeMem(pntr,int1)	P +	שיחרור שטח זיכרון שהוקצה בעזרת הפרוצדורה GetMem.
GetMem(pntr,int1)	P +	הקצאת שטח זיכרון לפי כמות בתים מבוקשת.
Mark(pntr)	P	שמירה של המצביע שבתחילת השטח החפשי בזיכרון, אשר מיועד להקצאה.
MaxAvail	F +	מחזירה את הגודל של שטח הזיכרון הרצוף הפנוי הגדול ביותר, מבין השטחים המיועדים להקצאה דינמית.
MemAvail	F +	מחזירה את גודל שטח הזיכרון שפנוי להקצאה דינמית.
New(pntr)	P	הקצאת שטח זיכרון למצביע, לפי גודל האיפיון שעליו הוא מצביע.
Pack(array)	P	דחיסת המערך כדי שיתפוס פחות שטח זיכרון. הפרוצדורה אינה קימת בטורבו-פסקל.
Ptr(int1)	F +	מחזירה ערך של מצביע, במערכת CP/M-80 בלבד.
Ptr(int1,int2)	F +	מחזירה ערך של מצביע, כלומר כתובת בזיכרון. int1 הוא הסגמנט בזיכרון ו-int2 הוא ההיסט מתחילת הסגמנט.
Release(pntr)	P +	שיחרור של שטח הזיכרון החל מהכתובת ש- pntr מצביע עליה. ערך זה נקבע לרוב ע"י הפרוצדורה Mark.
FillChar (memloc,int1,int2)	P +	הצבה של int1 תווים מסוג chr1, החל ממקום memloc בזיכרון.
Hi(int1)	F +	הפונקציה מחזירה את הערך של הבית הגבוה במשתנה (בטורבו-פסקל משתנה מאיפיון שלם תופס בזיכרון 2 בתים, אשר השמאלי שבהם הוא High=גבוה והימני הוא Low=נמוך).

F + Lo(int1)	מחזירה את ערכו של הבית הנמוך במשתנה. ראה גם הערה בפונק' Hi.
P + Move (memloc1,memloc2, int1)	העתקה של קטע בזיכרון של int1 בתים ממקום אחד למקום אחר בזיכרון.
F + Ofs(any)	מחזירה את כתובת ההיסט של המשתנה any מתחילת הסגמנט (הדף). הכתובת any יכולה להיות של משתנה, פ/פ או איפיון.
F + Seg(any)	מחזירה את מספר הסגמנט (הדף) שבו נמצא המשתנה any.
F + Cseg	מחזירה את הסגמנט בו נמצאת התכנית.
F + Dseg	מחזירה את הסגמנט בו נמצאים הנתונים.
F + Sseg	מחזירה את הסגמנט בו נמצאת המחסנית.
F + SizeOf(any)	מחזירה את גודל שטח הזיכרון שתופס משתנה או איפיון.
P + Swap(int1)	החלפה בין הבית הגבוה והנמוך שבמשתנה זה. משתנה מאיפיון שלם תופס בטורבו-פסקל 2 בתים בלבד.

<u>הערה:</u>

mark ו-release קיימים גם בניבי פסקל אחרים כמו למשל UCSD-Pascal, אך אינם חלק מפסקל סטנדרטי.

פרוצדורות ופונקציות לטיפול במסך ובלוח המקשים
Video and Keyboard Subprograms

P + ClrEol	ניקוי עד סוף השורה, בה נמצא הסמן (cursor).
P + ClrScr	ניקוי המסך כולו והעברת הסמן למצב Home.
P + CrtExit	Send terminal reset string
P + CrtInit	Send terminal initialization string
P + DelLine	מחיקת השורה שבה נמצא הסמן וצמצום רווח השורות (השורות "עולות").
P + GotoXY (int1,int2)	הצבת הסמן בטור int1 (החל משמאל), ובשורה int2 (מלמעלה). הפינה השמאלית העליונה של המסך נחשבת כנקודה (1,1).
P + HighVideo	הגברה של עצמת ההארה במסך (חד-צבע) עבור הפלט הבא.
P + InsLine	הכנסת שורה ריקה במסך, במקום בו נמצא הסמן. כל השורות שמתחתיו "יורדות".
F + IOresult	הפונקציה מחזירה ערך 0 אם פעולת הקלט/פלט האחרונה הצליחה, ומספר אחר - אם נכשלה. במקרה של שגיאה, הפונקציה מחזירה את מספר השגיאה שקרתה. יש להשתמש בפונקציה בשילוב עם ההוראה למהדר {$I-} (ראה גם בסעיף הוראות למהדר).
F + Keypressed	הפונקציה מחזירה true אם הקישו על מקש כלשהו, אחרת - מחזירה false.
P + LowVideo	החלשה של עצמת ההארה במסך, עבור הפלט הבא.
P + NormVideo	הגברה של עצמת ההארה של הפלט הבא (כמו HighVideo).
P - Page	פרוצדורה לדילוג לתחילת העמוד הבא במדפסת, או ניקוי המסך. פרוצדורה זו אינה מוגדרת בטורבו-פסקל.

פקודות לטיפול במסך לפי קוד ANSI
Video Control

n מספר תווים.
d כיוון התקדמות: 'A' - מעלה,
'B' - מטה,
'C' - שמאלה,
'D' - ימינה.

Write (Chr(27), '[', n, d)	הזזת הסמן ממקומו.
Write (Chr(27), '[s')	שמירת מיקום הסמן.
Write (Chr(27), '[u')	החזרת הסמן למיקום, אשר נשמר ע"י הפקודה הקודמת.
Write (Chr(27), '[2J')	ניקוי המסך כולו והעברת הסמן למצב Home.
Write (Chr(27), '[K')	ניקוי עד סוף השורה, בה נמצא הסמן.
Write (Chr(27), '[', x:1, ';', y:1 'H')	הצבת הסמן בשורה x (מלמעלה), ובטור y (משמאל). הפינה העליונה של המסך נחשבת כנקודה (1,1).
Write (Chr(27), '[0m')	חזרה למצב כתיבה של "לבן על גבי שחור". ביטול כל הפעולות הבאות.
Write (Chr(27), '[1m')	הגברה של עצמת ההארה של הפלט הבא.
Write (Chr(27), '[4m')	מתיחת קו מתחת לפלט הבא (במסך חד-צבע בלבד).
Write (Chr(27), '[5m')	היבהוב של הפלט הבא.
Write (Chr(27), '[7m')	היפוך בין צבע התו לצבע הרקע של הפלט הבא.
Write (Chr(27), '[8m')	הפלט הבא לא יראה.

הערה:

א. במחשבי IBM-PC הפועלים עם מערכת הפעלה DOS, יש לכלול בקובץ CONFIG.SYS את ההצבה הבאה:

device = [d:][path] ansi.sys

יש לדאוג לכך, שהקובץ ansi.sys אכן ימצא ב-path המתאים.

ב. אפשר לשלב יותר מאופן הצגה אחד של פלט על גבי המסך. למשל: הדגשה של הפלט בקו ובעצמת הארה.

ג. התו chr(27) הוא תו Escape, שמשמש לשליחת הוראות לאמצעי הפלט.

פ/פ לשליטה במערכת ההפעלה
System Control Subprograms

במערכת הפעלה Dos (MS-DOS או PC-DOS), קיימים הקבצים BDOS ו-BIOS, אשר מכילים פרוצדורות של מערכת ההפעלה לפעולות בסיסיות של המחשב כמו למשל, פעולות קלט/פלט. פונקציות ופרוצדורות אלו משמשות לקריאה והפעלה של השגרות הנמצאות בקבצים. השוני בין הפונקציות לפרוצדורות הוא בדרך שבה הן פועלות ובדרך שבה הם מחזירות ערכים.

P Bdos(int1[,int2])	CP/M-80: קריאה לשגרה מספר int1 של ה-Bdos, ואפשרות לשימוש בפרמטר נוסף int2.
F Bdos(int1[,int2])	CP/M-80: קריאה לשגרה מספר int1 של ה-Bdos, ואפשרות לשימוש בפרמטר נוסף int2. זוהי פונקציה, אשר מחזירה ערך באוגר A.
F BdosHL(int1[,int2])	CP/M-80: קריאה לשגרה מספר int1 של ה-Bdos, ואפשרות לשימוש בפרמטר נוסף int2. הפונקציה מחזירה את התוצאה בזוג האוגרים HL.
P Bdos(dos-reg)	CP/M-86: קריאה לשגרה של ה-Bdos, לפי הערכים שנמצאים הפרמטר dos-reg (ראה הערה).
P Bios(int1[,int2...])	CP/M-80: קריאה לשגרה מספר int1 של ה-Bios, ואפשרות לשימוש בפרמטרים נוספים.
F Bios(int1[,int2...])	CP/M-80: קריאה לשגרה מספר int1 של ה-Bios, ואפשרות לשימוש בפרמטרים נוספים. זוהי פונקציה, אשר מחזירה ערך באוגר A.
F BiosHL(int1[,int2...])	CP/M-80: קריאה לשגרה מספר int1 של ה-Bios, ואפשרות לשימוש בפרמטרים נוספים. הפונקציה מחזירה את התוצאה בזוג האוגרים HL.
P ChDir(str1)	MS-DOS (החל מגירסה 3): שינוי הספריה שפועלת בדיסק או בדיסקט הנוכחי, לספריה בשם str1 באותו דיסק או דיסקט.
P GetDir(int1,str1)	פרוצדורה שמחזירה לפרמטר int1 את מספר הכונן שאיתו עובדים, ולפרמטר str1 את שם הספריה שפועלים בה באותו כונן.
P Intr(dos-reg)	במערכות של 16 סיביות (IBM-PC למשל): יצירת interrupt לפי הערכים שמכיל הפרמטר (ראה הערה).

P MkDir(str1)	MS-DOS (החל מגירסה 3): יצירת תת-ספריה חדשה (sub-directory) בשם str1.
P MsDos(dos-reg)	קריאה למערכת ההפעלה MS-DOS בהתאם לפרמטר (ראה הערה מטה).
P OvrDrive(int1)	במערכות של 16 סיביות: הכרזה שקבצי ה-Overlay של פרוצדורות ופונקציות נמצאים בכונן מספר int1.
P OvrPath(str1)	MS-DOS (החל מגירסה 3): הכרזה שקבצי ה-Overlay של פרוצדורות ופונקציות נמצאים בכונן הנוכחי, בספריה בשם str1.
F ParmCount	הפונקציה מחזירה את מספר הפרמטרים במאגר של שורת הפקודה, שמוחזקת במערכת ההפעלה.
F ParmStr(int1)	הפונקציה מחזירה מחרוזת, שהיא ערך הפרמטר שמיקומו int1, במאגר של שורת הפקודה של מערכת ההפעלה.
P Delay(int1)	השעיית הביצוע של התכנית למשך זמן של int1 אלפיות שנייה.
P Halt	הפסקת ביצוע התכנית.
P Randomize	בשימוש החל מגירסה 3: בכל פעם שמורצת התכנית, המספרים האקראיים שמייצרת פונקצית ה-Random יהיו שונים, אחרת - פונקציה זו תיצור מספרים זהים בכל הרצה.
P RmDir(str1)	MS-DOS (החל מגירסה 3): ביטול הספריה שבדיסק או בדיסקט הנוכחי ששמה str1. הספריה תימחק רק אם היא ריקה מקבצים.

הערה:

הפרמטר dos-reg הינו רשומה שמכילה 10 שדות של שלמים, לפי המבנה של אוגרי המערכת.

```
dos-reg : record
        ax,bx,cx,dx,bp,si,di,ds,es,flags : integer; end;
```

פונקציות ופרוצדורות של טורבו-פסקל למחשב IBM-PC
IBM-PC System Subprograms

פ/פ הפועלות ללא כרטיס גרפי

P Delay(int1)	השהייה, דהיינו הפסקת ביצוע במחשב למשך זמן מוגבל. int1 - מספר אלפיות השניה של ההשהייה.
P NoSound	הפסקה של השמעת צליל.
P Sound(int1)	השמעת צליל לפי תדירות (לפי סקלה של הרץ). הצליל ישמע עד שתופעל הפרוצדורה NoSound.
P TextBackground(int1)	קביעת צבע הרקע במסך של תמליל.
P TextColor(int1)	קביעת צבע הצגת התווים במסך תמליל.
P TextMode	העברת המסך למצב של כתיבת תמליל, תוך שימוש במסך הנוכחי.
P TextMode(int1)	העברת המסך למצב של תמליל, תוך שימוש במסך מסוים.
F WhereX	מחזירה את מספר הטור שבו נמצא הסמן.
F WhereY	מחזירה את מספר השורה שבה נמצא הסמן.
P Window (int1,int2,int3,int4)	הגדרת חלון על המסך לפי הקואורדינטות של הפינה השמאלית העליונה (int1,int2) והפינה הימנית התחתונה (int3,int4) של החלון.

פ/פ לעבודה בכרטיס גרפי

P Arc (int1,int2,int3,int4, int5)	שרטוט קשת החל מנקודה (int1,int2), בזוית int3 מעלות, ברדיוס int4 ובצבע int5.
P Circle (int1,int2,int3,int4)	שרטוט של מעגל שמרכזו בנקודה (int1,int2), רדיוסו int3 וצבעו int4.
P ColorTable (int1,int2,int3,int4)	הגדרה של טבלת הסבה לצבעים int1 עד int4.
P Draw (int1,int2,int3,int4)	שרטוט קו ישר בין הנקודות (int1,int2) ו-(int3,int4).

P FillScreen(int1)	ניקוי המסך ע"י צביעתו בצבע int1.
P FillShape (int1,int2,int3,int4)	צביעת השטח מסביב לנקודה (int1,int2) בצבע int3. גבולות שטח הצורה מותווים בצבע int4.
P FillPattern (int1,int2,int3,int4, int5)	צביעת מלבן שמוגדר בין הנקודות (int1,int2) ו-(int3,int4) בצבע int5.
F GetDotColor (int1,int2)	מספר הצבע של הנקודה (int1,int2).
P GetPic (any,int1,int2,int3, int4)	החסנת הגרפיקה שבתחום שבין הנקודה השמאלית-עליונה (int1,int2) לבין הנקודה הימנית-תחתונה (int3,int4) אל המאגר המוגדר ע"י any.
P GraphBackground(int1)	בחירת צבע הרקע במסך הגרפי.
P GraphColorMode	הפעלת מסך גרפי בהפרדה של 320x200 נקודות. הצבעים שיוצגו על המסך תלויים בפרוצדורה Palette.
P GraphWindow (int1,int2,int3,int4)	הגדרת חלון במסך גרפי לפי נקודה שמאלית-עליונה (int1,int2) ו-נקודה ימנית-תחתונה (int3,int4).
P HiRes	הגדרת הפרדה (רזולוציה) של 640x200 נקודות (בד"כ מתאים רק למסך חד-צבע).
P HiResColor(int1)	כמו הפונקציה HiRes. כאן ניתן בנוסף, לבחור את הצבע בו יתבצע השרטוט במסך.
P Palette(int1)	הגדרה בהפרדה של 320x200 נקודות, ובחירת צירוף הצבעים שבו תתבצע התצוגה.
P Pattern(pntr)	הגדרת התבנית (pattern) שבה ישתמש המחשב בפונקציה FillPattern. pntr הוא מערך של 7 בתים, אשר בו נתונים לגבי סוג התבנית.
P Plot(int1,int2,int3)	קביעת הצבע (int3) של הנקודה (int1,int2).
P PutPic(any,int1,int2)	הצגה של גרפיקה, שנשמרה קודם לכן במשתנה any ע"י הפרוצדורה GetPic, ימינה ומתחת לנקודה (int1,int2) שעל המסך.

<u>פקודות ל"צב" במחשב IBM-PC</u>
<u>IBM-PC Turtle Graphics</u>
(לביצוע פקודות אילו דרוש כרטיס גרפי)

P	Back(int1)	הזזת ה"צב" אחורה, int1 צעדים.
P	ClearScreen	ניקוי החלון במסך בו פועל ה"צב", ומיקומו בנקודה 0,0.
P	Forward(int1)	הזזת ה"צב" קדימה ב-int1 צעדים.
F	Heading	החזרה של כיוון תנועת ה"צב" במעלות.
P	HideTurtle	הפיכת ה"צב" לבלתי-נראה.
P	Home	מיקום ה"צב" במרכז החלון בו הוא פועל (נקודה 0,0).
P	NoWrap	מונע מה"צב" להופיע בצידו השני של החלון, כאשר הוא זז מחוץ לגבולות החלון בצידו האחד.
P	PenDown	גורמת לכך שכל תזוזה של ה"צב" תשורטט על המסך.
P	PenUp	גורמת לכך שה"צב" יזוז מבלי לשרטט.
P	SetHeading(int1)	קביעת כיוון ה"צב" במעלות (0 מעלות=למעלה, 180 מעלות=למטה).
P	SetPenColor(int1)	קביעת הצבע שבו הצב משרטט.
P	ShowTurtle	להצגת ה"צב" (ההיפך מהפרוצדורה HideTurtle).
P	TurnLeft(int1)	סיבוב ה"צב" ב-int1 מעלות שמאלה (נגד כיוון השעון).
P	TurnRight(int1)	סיבוב ה"צב" ב-int1 מעלות ימינה (עם כיוון השעון).
P	TurtleWindow (int1,int2,int3,int4)	הגדרת החלון בו פועל ה"צב", כאשר: מרכז החלון היא הנקודה (int1,int2), רוחבו int3, וגובהו int4.
F	TurtleThere	קבלת ערך True אם ה"צב" נמצא בחלון הנוכחי שלו.
P	Wrap	ה"צב" יראה מצידו השני של החלון, כאשר הוא יוצא מגבולות החלון בצידו האחד.

F Xcor קורדינטת ה-X שבה נמצא ה"צב".

F Ycor קורדינטת ה-Y שבה נמצא ה"צב".

משתנים בשרות המערכת
System Level Variables

BufLen כמות התווים המכסימלית שאפשר לקלוט בפרוצדורת ה-Read.

HeapPtr מצביע לראש המאגר שמשמש להקצאה דינמית של שטחי זיכרון.

RecurPtr מצביע לראש המחסנית שמשמשת לרקורסיה של פונקציות ופרוצדורות.

StackPtr מצביע לראש המחסנית שמשמשת להקצאת שטחים למשתנים מקומיים בזמן קריאה לפונקציות ופרוצדורות.

מצביעים לכתובות של פרוצדורות קלט/פלט
I/O Driver Vector Address Variables
(טורבו-פסקל בלבד)

כל המשתנים שלמטה הינם משתנים סטנדרטיים בטורבו-פסקל (PreDefined). הם מצביעים על מיקומן של שגרות קלט או פלט של אמצעי קלט/פלט סטנדרטיים. הדבר שימושי, כדי לאפשר למשתמש להגדיר אמצעי קלט/פלט כרצוננו.
יש לשים לב לכך, שאין אלו פ/פ סטנדרטיות, כלומר לא ניתן לקרוא להן כפי שקוראים לכל פ/פ אחרת.
לבירור נוסף כיצד מוגדר כל אמצעי שמפורט מטה, הסתכל בסעיף של קבצי תמליל סטנדרטיים.

שם המצביע	מצביע על
AuxInPtr	מצביע לפרוצדורה Aux לקלט.
AuxOutPtr	מצביע לפרוצדורה Aux לפלט.
ConInPtr	מצביע לפרוצדורה Con לקלט.
ConOutPtr	מצביע לפרוצדורה Con לפלט.
ConStPtr	מצביע לפונקציה המחזירה את מצב Con.
ConLstPtr	מצביע לפרוצדורה Lst לפלט.
UsrOutPtr	מצביע לפרוצדורה Usr לקלט.
UsrOutPtr	מצביע לפרוצדורה Usr לפלט.

מערכים סטנדרטיים
Memory and Port Array Variables

Mem[ ] מערך של איברים מאיפיון בית (byte), למיפוי של כל הזיכרון. במעבדים 8086 ו-8088, האינדקס הראשון של המערך הוא מספר הסגמנט (הדף) בזיכרון, והאינדקס השני הוא ההיסט של הבית מתחילת הסגמנט. במעבד Z-80 האינדקס הוא מספר שלם יחיד.

MemW[ ] מערך של מלים (Words) בנות 16 סיביות, למיפוי של כל הזיכרון. מערך זה קיים רק במחשבים של 16 סיביות. האינדקסים של המערך הם כמו של המערך Mem (לא קיים במעבד Z-80).

Port[ ] מערך של איברים מאיפיון בית, למיפוי של הכניסות והיציאות (לקלט ופלט) של המחשב. למערך יש אינדקס אחד שערכו יכול להיות בטווח מ-0 ועד 65535 (תיאורטית ניתן לחבר למחשב 65536 יחידות קלט/פלט). כל איבר פועל ככניסה/יציאה טורית. במעבד Z-80 אינדקס המערך הוא מאיפיון byte וכן גם איברי המערך.

PortW[ ] מערך שאיבריו הן מלים (Words) בנות 16 סיביות. מערך זה פועל במחשבים של 16 סיביות בלבד. תפקידו ואופן פעולתו דומים לאלו של המערך Port (לא קיים המעבד Z-80).

הגדרות מראש לאמצעי קלט/פלט
Text File Predeclared Variables

+ Aux פלט ליציאה הטורית של המחשב. בד"כ יכול לשמש למודם או למדפסת בעלת תקשורת טורית.

+ Con פלט למסך המחשב.

Input אמצעי הקלט, שהוא ברירת המחדל של המערכת, בד"כ - המקלדת.

+ Kbd מקלדת - אמצעי קלט בלבד.

+.Lst המדפסת - אמצעי פלט בלבד.

Output אמצעי הפלט, שהוא ברירת המחדל של המערכת, בד"כ - המסך.

+ Trm פלט למסוף של המערכת. בד"כ יש זהות ל-Con.

+ Usr אמצעי לא מוגדר במערכת. המתכנת יכול להגדירו ע"י פעולת Assign.

סימנים מיוחדים
Special Symbols

# +	מקדים מספר שלם בתחום 0-255, כדי לציין תו לפי קוד ASCII,או תו מיוחד.
$ +	מציין שאחריו מופיע קבוע בבסיס 16.
' (תג)	תג יחיד מציין תחילתה וסופה של רשימת קבועים מסוג מחרוזת, או תו.
()	הסוגריים תוחמים רשימת פרמטרים של פ/פ. הסוגריים משנים את סדר הביצוע של פעולות בביטויים מאיפיונים שונים. בטורבו-פסקל משמש גם בהגדרת קבוע מאופיין (typed constants).
,	הפסיק מפריד בין פרמטרים בהגדרה וקריאה לפ/פ, ובין משתנים בחלקים Var, Type ו-Label של התוכנית.
.	הנקודה מסמנת סוף תכנית. היא משמשת גם בתוך קבועים ממשיים (כדי לציין את חלק השבר), ולרישום שדות של רשומה.
..	שתי נקודות מציינות תת-תחום של איפיון מסוים.
:	הנקודותיים מפרידות בחלק ה-type בין שם האיפיון החדש והגדרתו. בהצהרת Var, וברשימת פרמטרים של פ/פ - בין משתנים והאיפיון שלהם.
:=	סימון משולב זה משמש בפקודת הצבה/השמה.
;	סימן להפרדה בין פקודות ולהפרדה בין הצהרות הגדרות וכותרות.
=	סימן של יחס בין ביטויים במשפט בוליאני. הוא מפריד בין הקבוע וערכו בהגדרה של קבועים (const).
[] או (..)	סימון לתת-תחום שנמצא בתוך קבוצה.
^	מגדיר ומציין איפיון מסוג מצביע, ומקדים את האותיות והתווים], \, [, ^, - כדי לציין תו בקרה.
{} או (* *)	תוחמים של הערות בגוף התוכנית.

הוראות למהדר טורבו-פסקל
Turbo-Pascal Compiler Directives

{$A+} עבור מערכת CP/M-80: +, איסור לבצע רקורסיות; -, היתר לבצע רקורסיות.

{$B+} אם -, הקלט/פלט מתייחס לאמצעי המוגדר :TRM; +, אמצעי הקלט/פלט הוא :CON.

{$C+} אם +, מאפשר קבלת הפרעה (Intrrupt) של Ctrl-S או Ctrl-C בזמן הרצה; -, אינו מאפשר זאת.

{$D+} אם +, מאפשר מאגר (Buffer) של קלט/פלט; -, אינו מאפשר.

{$Fn} n הוא מספר הקבצים שניתן לפתוח בו-זמנית.

{$Gn} אם n=0 אזי אמצעי הקלט הסטנדרטי יהיה CON או TRM לפי ההוראה {B+}; אחרת, יש אפשרות לקבוע את אמצעי הקלט ואת גודל המאגר. הוראה זו קיימת במערכת הפעלה MS-DOS בלבד.

{$I+} אם +, הטיפול בשגיאות קלפ/פלט יהיה ע"י המערכת; אחרת, הטיפול בשגיאות קלט/פלט יהיה ע"י המתכנת, אשר יזהה אותן בעזרת הפונקציה IOresult.

{$I file} קובץ מקור של פ/פ שהוכן בנפרד, יהיה כלול בתכנית.

{$K+} אם +, נעשית בדיקה אם יש במחסנית מספיק שטח למשתנים המקומיים של הפ/פ; אם -, לא נעשית בדיקה כזו והמשמעות היא - ביטול הבדיקה של גלישה במחסנית.

{$O@} מציין שקבצי OVERLAY (של פונקציות ופרוצדורות) נמצאות על תקליטון בכונן שהוא ברירת המחדל; אחרת, נחליף את התו @ באחד התווים בתחום A..P, שיציין את שם הכונן שבו הם נמצאים.

{$Pn} הוראה זו מגדירה את הפלט הסטנדרטי, בדומה להוראה {$Gn} אשר מגדירה את הקלט הסטנדרטי.

{$R-} אם +, תתבצע בזמן הרצה בדיקה אם אינדקס של מערך אינו מחוץ לטווח ההגדרה של המערך; אם -, בדיקה כזו לא תתבצע (עלולות להיות שגיאות תכנות).

{$V+} אם +, תתבצע בדיקה בעת השוואה בין מחרוזות, אם אורכן שווה; אם -, תבוטל הבדיקה.

{$U-} אם +, אין אפשרות לקלוט פסקים interupts מסוג Ctrl-C. הדבר מגביר את מהירות הרצת התכנית; אם -, יש לפעול לפי ההוראה {$C+}.

{$Wn} ציון של רמת הקינון המכסימלית של הפרוצדורה WITH. n יכול לקבל ערכים מ-1 עד 9. ברירת המחדל היא 2.

{$X+} אם +, בעיבוד מערכים תתבצע אופטימיזציה כדי לשפר את הביצועים. אם -, מתבצעת אופטימיזציה כדי להקטין את גודל התוכנית.

<u>הערה:</u>

הערך הרשום ליד ההוראה הוא ברירת המחדל.



לומדים מחשבים — בסיומו של יום־עיון

אינדקס ומילון מונחים

האינדקס שלהלן כולל את הנושאים העיקריים, אשר נדונו בשאלות ובתרגילי התכנות שבספר זה. לנוחות הלומד נרשמו מספרי התרגילים שבהם יושם הנושא בהרחבה. באינדקס משולב גם מילון, והוא ניתן במיון כפול - לפי העברית ולפי האנגלית.

את מראי המקומות יש לקרוא משמאל לימין, כך:

לדוגמה: 7 פרושו פרק 7 — פרק
7_2 פרושו פרק 7 שאלה 2 — שאלה_פרק
7/2 פרושו פרק 7 תוכנית 2 — תוכנית/פרק

רשימות

ספר התשובות לפסקל מציג בפני הקורא מיגוון של תוכניות פסקל העוסקות במספר רב של נושאים למתחילים ומתקדמים.
אלו הן תשובות לשאלות חזרה ותרגילי תכנות שבספר **פסקל — שפה לתכנות מבני** מאת פרופ׳ אבא אנגלברג.

התוכניות שבספר מלוות בהסברים מפורטים למשתנים ולמבני הקבצים והרשימות למיניהן. תרשימים סכימטיים מתארים את מהלך התוכניות, את הפרוצדורות והפונקציות ותפקידן, ואת מבנה הקבצים והרשימות.

הסבר מקיף ניתן לתוכניות העוסקות ב**מבני נתונים דינמיים** ו**תוכניות מיון**.

פרק נוסף מתאר את הגדרות מהדר פסקל הסטנדרטי, עם דגש על מהדר **Turbo-Pascal 3.0**.

הספר מיועד לכל הלומדים פסקל לפי הדרישות של תוכנית הלימודים החדשה של משרד החינוך והתרבות עבור מגמות ענ״א בבתי הספר העל־יסודיים ובמכללות להנדסאים. הוא מותאם גם למיגוון הנושאים הנלמדים במרכז ענ״א במכון לפריון העבודה והייצור, המכון להכשרה טכנולוגית (מה״ט) ולתוכניות הלימודים באוניברסיטאות ובמכללות למחשבים.

הספר מיועד לתלמיד בכתה, ללומד בעצמו, ולכל מי שכבר מתכנת פסקל ומעוניין להעמיק את ידיעותיו.

על המחבר:
אמיר קפלן נולד ב-1961, בישראל, למד מתמטיקה ומחשבים באוניברסיטת תל־אביב, חבר סגל בפקולטה למחשבים בטכניון. עסק בחברת סייטקס בנושאי יישום ופיתוח.